Studien zur Wissenschafts- und Technikforschung
Herausgegeben von Prof. Dr. Wolfgang Krohn
und Prof. Dr. Peter Weingart

Heike Stach

Zwischen Organismus und Notation

Zur kulturellen Konstruktion des Computer-Programms

DUV

Heike Stach
Zwischen Organismus und Notation

DUV Sozialwissenschaft

Studien zur Wissenschafts- und Technikforschung
Herausgegeben von Prof. Dr. Wolfgang Krohn
und Prof. Dr. Peter Weingart

Die

Studien zur Wissenschafts- und Technikforschung

richten den Blick auf die Funktionen von Forschung und technologischer Entwicklung in der entstehenden Wissensgesellschaft. Wissenschaft und Technik sind einerseits die wichtigsten Garanten der Innovationsfähigkeit der Gesellschaft, andererseits aber auch Quellen neuer Unsicherheiten und Befürchtungen.
In den Bänden der Schriftenreihe werden neue Formen der Wissenserzeugung, die Bewältigung von Risiken sowie die Konflikte zwischen unterschiedlichen Interessen und Wissenskulturen analysiert.

Heike Stach

Zwischen Organismus und Notation

Zur kulturellen Konstruktion des Computer-Programms

Mit einem Geleitwort von Prof. Dr. Ralf Bohnsack

Deutscher Universitäts-Verlag

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Stach, Heike:
Zwischen Organismus und Notation : zur kulturellen Konstruktion des Computer-Programms / Heike Stach. Mit einem Geleitw. von Ralf Bohnsack. – 1. Aufl.. – Wiesbaden : Dt. Univ.-Verl., 2001
(Studien zur Wissenschafts- und Technikforschung)
(DUV : Sozialwissenschaft)
Zugl.: Berlin, Techn. Univ., Diss. 2000
ISBN-13: 978-3-8244-4433-5 e-ISBN-13: 978-3-322-86898-5
DOI: 10.1007/978-3-322-86898-5

D 83

1. Auflage Februar 2001

Alle Rechte vorbehalten
© Deutscher Universitäts-Verlag GmbH, Wiesbaden, 2001

Lektorat: Ute Wrasmann / Gereon Roeseling

Der Deutsche Universitäts-Verlag ist ein Unternehmen der Fachverlagsgruppe BertelsmannSpringer.

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

www.duv.de

Höchste inhaltliche und technische Qualität unserer Produkte ist unser Ziel. Bei der Produktion und Verbreitung unserer Bücher wollen wir die Umwelt schonen. Dieses Buch ist deshalb auf säurefreiem und chlorfrei gebleichtem Papier gedruckt. Die Einschweißfolie besteht aus Polyäthylen und damit aus organischen Grundstoffen, die weder bei der Herstellung noch bei der Verbrennung Schadstoffe freisetzen.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

ISBN-13: 978-3-8244-4433-5

Geleitwort

Technische Artefakte, unter denen die Maschinen der Computertechnologie wohl die faszinierendsten sind, werden von Menschen einer bestimmten Zeit und Kultur hergestellt. Dennoch werden sie in aller Regel nicht als Ausdruck oder Dokument der Weltanschauung einer Epoche, einer zeitgeschichtlichen Phase oder eines Milieus verstanden, wie dies z.B. bei Werken oder Ideen der bildenden Kunst, der Geistesgeschichte oder der Sozialwissenschaften eher selbstverständlich ist. Auch die Texte, die sich um eine Technik ranken bzw. diese begründen, technisch-wissenschaftliche Texte also, werden als objektive Beschreibungen in dem Sinne genommen, daß sie jenseits einer kulturellen Standortgebundenheit anzusiedeln sind. Und tatsächlich zeichnen sich solche Texte dadurch aus, daß sie von den Spuren ihrer sozialen Herkunft, den persönlichen Anmerkungen ihrer Autoren oder gar persönlichen Kommentierungen weitgehend bereinigt sind.

Aber dennoch sind auch diese Texte – und dies kann Heike Stach in überzeugender und Schritt für Schritt nachvollziehbarer Weise zeigen – einer Interpretation zugänglich, die nach deren kultureller Gewordenheit fragt. Der expliziten Darstellung und Begründung von Theorien oder Techniken liegt ein – um es mit Michael Polanyi auszudrücken – implizites oder stillschweigendes Wissen („tacit knowledge") zugrunde. Mit diesem impliziten Wissen werden kulturell und zeitlich gebundene Orientierungsmuster transportiert. Karl Mannheim, auf dessen „dokumentarische Methode" das von Heike Stach angewandte Interpretationsverfahren ursprünglich zurückgeht, spricht hier von einem „atheoretischen Wissen".

Sich so weitgehend auf interdisziplinäres und somit ungesichertes Terrain zu wagen wie Heike Stach, ist ungewöhnlich und äußerst mutig. Daß sie sich hier ihren Weg sicher zu erschließen weiß, zeugt zugleich von hoher Kompetenz nicht nur im Feld der Informatik, sondern auch in demjenigen der sozial- und kulturwissenschaftlichen Methodologie. Sie erreicht damit ein hohes Maß an analytischer Distanz gegenüber dem fachspezifischen Diskurs der Informatik und vermag auf diese Weise neue Perspektiven zu eröffnen: Der mit solchen Interpretationen verbundene Perspektivenwechsel kann einen Beitrag leisten, den Methodenkanon der Informatik – und vielleicht auch den anderer Technikwissenschaften – zu erweitern und zu bereichern. Schließlich und nicht zuletzt kann diese Arbeit deutlich machen, daß es die inhaltlichen Ergebnisse einer solchen Interpretation ermöglichen, Kontinuität und Veränderung innerhalb der Technik- und Wissenschaftsentwicklung zu begreifen als Ausdruck der Konstanz oder des Wandels von unterliegenden, d.h. impliziten oder atheoretischen Orientierungsstrukturen der Akteure und der sie tragenden Gruppen oder Milieus.

Zentrale inhaltliche Ergebnisse, die die Autorin durch ihre detaillierte und methodisch-systematische Analyse zentraler Texte aus der Entwicklung der Computertechnologie und Programmierung zu Tage fördert, lassen sich durch folgende Stichworte umreissen: Computer und Programme sind in ihrer Entwicklung eingebettet in implizite kulturelle Wissensbestände, d.h. in kulturelle Strömungen. Sie erscheinen zugleich als deren Ausdruck wie als deren Triebfeder. Zum Zeitpunkt der Entstehung der Computer und ihrer Programme zeichnet sich die hier ausschlaggebende kulturelle Strömung dadurch aus, daß die Differenz natürlich-technisch sowie Organismus-Symbolsystem in Frage gestellt wurde. Pate steht dabei eine enggeführte Handlungstheorie nach Art des Behaviorismus, der Handeln auf Reiz-Reaktionsketten reduziert. Die Rekonstruktion dieses, von Heike Stach als „Hybridobjekt-Sichtweise" bezeichneten Orientierungsmusters bildet den roten Faden der Arbeit. Zugleich gibt die Autorin jedoch auch der Spezifik der verschiedenen Technologieentwicklungen Raum, indem deren Unterschiede durch vergleichende Analysen herausgearbeitet werden.

Die untersuchten Fachtexte werden auch hinsichtlich ihrer je individuellen Entstehungsgeschichte analysiert. Diese ist sorgfältig recherchiert, u.a. auch auf der Grundlage von Interviews mit dem jeweiligen Autor, und wird auch in den Kontext der technologischen Entwicklung gestellt. Beispielsweise gibt die Autorin einen Überblick über den jeweiligen Stand der technologischen Entwicklung im Bereich der Computertechnologie und Programmierung, in den sie auch die Rezeption dieser Entwicklung in der Tagespresse einbezieht.

Aufgrund des interdisziplinären Charakters, durch den das gesamte Buch geprägt ist, ist die Lektüre sowohl für Informatiker und EDV-Interessierte, die mehr über die Geschichte der Programmierung und die Einbettung dieser Pionierleistungen in die Weltanschauung ihrer Entstehungsphase erfahren wollen, eine Bereicherung, als auch für kulturwissenschaftlich und (technik-)soziologisch Interessierte. Heike Stach hat mit diesem Buch nicht nur vom sachlichen Ertrag ihrer Interpretationen her eine außergewöhnliche interdisziplinäre Leistung vollbracht. Ihr ist es auch gelungen, in einem Stil zu schreiben, der über die Grenzen der Disziplinen hinweg verständlich ist.

Prof. Dr. Ralf Bohnsack

Inhalt

Einleitung 1

Kapitel 1: Wissenschafts- und Technikentwicklung als Denk- und Handlungspraxis 8

1.1 Implizites Wissen in der Wissenschafts- und Technikentwicklung 9

1.2 Dokumentarische Interpretation technischer Texte 19

Kapitel 2: Die Entstehung der Speicherprogrammierung 30

2.1 Vorgeschichte: Von Neumann, das ENIAC- und das EDVAC-Projekt 32

2.2. Text: „First Draft of a Report on the EDVAC“ 44

2.2.1. Technischer Diskurs 46

2.2.2 Neurophysiologischer Diskurs 49

Exkurs: McCulloch/Pitts und Turing 51

2.2.3 Mathematischer Diskurs 55

2.2.4 Das Rechengerät: Maschine – Organismus – naturgegebene Struktur 57

2.2.5 Code 59

2.3 Erste speicherprogrammierbare Rechenmaschinen 63

2.4 Die IAS-Maschine 66

2.4.1 Text: „Preliminary Discussion of the Logical Design of an Electronic Computing Instrument“ 69

2.4.2 Text: „Planning and Coding Problems for an Electronic Computing Instrument“ 74

2.5 Zusammenfassung: Die Speicherprogrammierung zwischen technischem Mechanismus und mathematischer Anweisung 84

Kapitel 3: Immaterielle Maschinen 89

3.1 Text: „The General and Logical Theory of Automata“ 92

3.1.1 Objekte als Beschreibung 94

Exkurs: Von der behavioristischen Untersuchung zum axiomatischen Vorgehen 97

3.1.2 Materielle Beschreibungen 104

3.2 Automaten und Programme 107

3.2.1 Instruktionen und Gene 109

3.2.2 Programmieren als behavioristisch-atomistische Technik 112

3.3 Zusammenfassung: Mensch – Maschine – Symbolsystem 114

Kapitel 4: Programmiersprachen 121
4.1 Herstellbare Sprachen 123
4.2 Wissenschaft und Computereinsatz in den USA der Nachkriegszeit 126
4.3 Kodiersysteme 133
4.4 Das „FORmula TRANslating System“ 136
4.4.1 Vorgeschichte 137
4.4.2 Text: „The FORTRAN Automatic Coding System“ 142
4.4.2.1 Adressaten und Selbstpräsentation 143
4.4.2.2 Der Kodierer und die Maschine 144
4.4.2.3 Das FORTRAN-System 146
4.4.2.3.1 Die FORTRAN-Sprache 147
4.4.2.3.2 Der FORTRAN-Übersetzer 154
4.4.3 Zusammenfassung: FORTRAN als automatisches Kodiersystem 158
4.5 Maschinenunabhängige Programmiersprachen 162
4.5.1 Die „ALGOrithmic Language“ 164
4.5.2 Die „COmmon Business (O) Language“ 166
4.6 Die „Information Processing Language“ und der „LISt Processor“ 169
4.6.1 IPL 173
4.6.1.1 Vorgeschichte 174
4.6.1.2 Der „Logic Theorist“ 176
4.6.1.3 IPL – die Sprache 180
4.6.1.4 Zusammenfassung: Die Rechenmaschine als behavioristisches Labor 185
4.6.2 LISP 187
4.6.2.1 Vorgeschichte 188
4.6.2.2 Der „Advice Taker“ 191
4.6.2.3 LISP – die Sprache 194
4.6.2.4 Zusammenfassung: LISP als universelle Sprache 199
4.7 Sprache und Programmiersprachen 202

Kapitel 5: Schlüsse 206
5.1 Hybridobjekte und die behavioristisch-atomistischen Sichtweise in der Programmierung 206
5.2 Computer und Programmierung heute 214
5.3 Die Explikation von Orientierungsmustern in der aktuellen Wissenschafts- und Technikentwicklung 223

Literatur 228

Einleitung

Diese Arbeit widmet sich der Entstehung von Welten, die aus nichts als Zeichen zusammengesetzt sind. Gemeint sind durch Programmieren geschöpfte „Universen von möglicherweise unbegrenzter Komplexität", wie Weizenbaum 1977 formulierte. Die hier angesprochene Vorstellung, Programmieren habe mit der Konstruktion von auch gegenständlich Gedachtem zu tun, war auch virulent, als im deutschsprachigen Raum versucht wurde, die Informatik als Wissenschaft und damit verbunden die Tätigkeit des Programmierens genauer zu fassen. So sprach Zemanek, einer der Begründer der österreichischen Informatik, von dem Informatiker als einem „Ingenieur abstrakter Objekte" und stellte fest (Zemanek 1971, S. 160f.): „Beim Informatiker sind die Gebilde, über die er spricht, bereits abstrakt und auf dem Papier, nämlich Programme und Beschreibungen [...]. Der Informatiker konstruiert, aber was er konstruiert sind abstrakte Objekte." Programmieren stellt sich also für Zemanek als Konstruktion von „Objekten" auf dem Papier, als Konstruktion durch (Be)schreiben dar – ein Bild, das Bauer, ein entscheidender Protagonist der bundesdeutschen Informatik, 1974 auf den Punkt brachte, als er die Informatik als „Ingenieur-Geisteswissenschaft (oder eine Geistes-Ingenieurwissenschaft, wem das besser gefällt)" (Bauer 1974, S. 336) charakterisierte. Daß im Diskurs der Programmierung die Differenz zwischen einer räumlich-gegenständlichen und einer in Zeichen aufgehenden, simulierten und programmierten Welt verschwimmt, wird in der aktuellen Debatte um „Cyberspace" und virtuelle Welten, um Besuche im Weißen Haus über das World Wide Web, Surfen und Sex im Internet besonders deutlich.

Wie kam es, so meine übergeordnete Frage, zu dieser Verräumlichung, Vergegenständlichung, Verweltlichung der Programmierung und ihrer Produkte? Woher, so könnte man allgemeiner formulieren, diese verblüffende Vermischung von Konstruktion und Deskription, Gegenstand und Zeichen, die für den Diskurs um die Programmierung selbstverständlich geworden ist? Auf der Suche nach einer Antwort frage ich in dieser Arbeit nach den Sinnstrukturen, die dieser Wahrnehmung der Programmierung und ihrer Produkte zugrunde liegen – und damit vielen der Visionen, technisch-wissenschaftlichen Entwürfe und Projekte in diesem Bereich. Angeregt durch die neuere sozialwissenschaftliche und historische Wissenschafts- und Technikforschung gehe ich dabei davon aus, „daß nichts in der Wissenschaft Sinn ergibt ohne eine Rekonstruktion der Geschichten, die sie erzählt." (Rheinberger et al. 1997, S. 11). Um die heutige Wahrnehmung der Programmierung besser zu verstehen, werde ich zunächst Geschichten rekonstruieren, die am Anfang der Speicherprogrammierung (Kapitel 2 und 3) und circa zehn Jahre später bei der Herausbildung der ersten Programmiersprachen (Kapitel 4) Pate standen. Abschließend suche ich nach den Spuren dieser Geschichten in der neueren Debatte um die Programmierung (Kapitel 5).

Angelehnt an Methoden aus der rekonstruktiven Sozialforschung, die ich – wie auch meinen analytischen Standpunkt – in Kapitel 1 genauer darstelle, frage ich danach, wie die im Entstehen begriffenen Techniken interpretiert wurden, und welche Anschauungen dabei eine Rolle spielten: Vor dem Hintergrund welcher allgemeineren und zumeist nicht explizit ausformulierten Weltanschauungen bildeten sich Ideen, wurden bestimmte zweckrationale Argumentationen akzeptiert und in wissenschaftliche Theorie und Technik umgesetzt? Dabei wird sich zeigen, daß die rekonstruierten Geschichten sich zwar im Laufe der Zeit verändern, aber doch von Beginn an von dem Aneinanderrücken materieller und immaterieller Welten und von Gegenstand und Zeichen erzählen.

Wie schon die vorangegangenen Überlegungen deutlich machen, geht es in dieser Arbeit nicht darum, einzelne technisch-wissenschaftliche Leistungen herauszustellen und rückblickend zu bewerten. Diese Arbeit haben andere vor mir geleistet. Vielmehr geht es darum, wie die einzelnen Projekte im Kontext ihrer Zeit denkbar wurden, wie sie interpretiert wurden, und wie diese Interpretationen und Wahrnehmungen sich im Laufe der Zeit tradierten und veränderten. Dieses Anliegen hat Folgen für die Art der Darstellung, die ich in dieser Arbeit wähle: Um die verschiedenen Wahrnehmungsarten der Rechenmaschine, der Programmierung und von Programmen möglichst authentisch wiederzugeben, werde ich bei der Besprechung der einzelnen Entwicklungen weitgehend auf die Terminologie der Zeit zurückgreifen und darauf verzichten, sie mit Hilfe heutiger Begrifflichkeit zu beschreiben. Das wirkt teilweise etwas befremdlich, ist jedoch für die Rekonstruktion des damaligen Verständnisses und der Sinnzusammenhänge, in die die entstehenden Artefakte eingebettet wurden, unerläßlich. Für meine Rekonstruktionen greife ich zum Teil auf Interviews mit beteiligten Akteuren[1], besonders aber auf in den verschiedenen Stadien verfaßte und breit rezipierte technische und wissenschaftliche Texte zurück. Diese Dokumente untersuche ich daraufhin, wie die dort dargestellten technisch-wissenschaftlichen Entwürfe und zweckrationalen Entscheidungen – in der von mir gewählten Terminologie „Orientierungsschemata" – mit impliziten Wissensbeständen – dem „Orientierungsrahmen" – zusammenhängen; ich erarbeite die den Texten unterliegenden „Orientierungsmuster", die Orientierungsschema und -rahmen umfassen.[2] Transparenter wird so das kulturelle

---

1 Diese Interviews verwende ich vor allem dazu, die jeweiligen Ereignisse lebendiger darzustellen. Für die Erarbeitung von der jeweiligen technischen Entwicklung unterliegenden Sinngeflechten können sie nur eine sekundäre Rolle spielen. Denn die Erzählungen der einzelnen Personen sind durch die zwischenzeitlich erfolgte technisch-wissenschaftliche Entwicklung, die damit verbundene Begrifflichkeit und im Nachhinein entstandene Erklärungsmuster überformt. Sie geben deshalb über die ursprünglichen Sinnstrukturen und Wahrnehmungen nur bedingt Auskunft.

2 Die Begriffe Orientierungsschema und -rahmen sowie den des Orientierungsmusters führe ich in Kapitel 1 ein. Die Begriffe – so wie ich sie verwende – stammen aus der rekonstruktiven Sozialforschung und gehen auf Bohnsack (1998) zurück.

Deutungsgefüge, in welches die einzelnen Entscheidungen eingebettet waren, und die Art und Weise, wie diese Deutungen[3] die Gestalt der entstehenden wissenschaftlichen und technischen Produkte ausformten. Umgekehrt frage ich danach, wie die so interpretierten technischen Artefakte – die Rechenmaschine, der Code, Programme und Programmiersprachen – zur Deutung und Erklärung sozialer und natürlicher Prozesse herangezogen wurden, und danach, ob und wie diese Erklärungen und Sichtweisen ihrerseits für die folgende Technikentwicklung relevant waren.

Jenseits aller Unterschiede, die aufgrund der unterschiedlichen Entstehungszeiten und Zielsetzungen nicht verwundern, schälte die Analyse der genannten Dokumente auch ein gemeinsames Orientierungsmuster heraus: Oben habe ich bereits bemerkt, daß die rekonstruierten Geschichten von dem Verschwimmen materieller und immaterieller Welten und von Gegenstand und Zeichen erzählen. Dieses Phänomen ist im interdisziplinären Forschungsprojekt „Sozialgeschichte der Informatik"[4], in dessen Rahmen ich diese Arbeit geschrieben habe, auch in anderen als den hier besprochenen Dokumenten gefunden worden: so besonders in Texten zur Rechenautomatenforschung und der Herausbildung des Hochschulstudiengangs Informatik in der Bundesrepublik Deutschland.[5] Die verschiedenen Autorinnen und Autoren behandeln oder kreieren, ohne daß dies theoretisiert oder expliziert wird, eine besondere Klasse von „Gegenständen", die wir, um dieses Phänomen begrifflich zu fassen, Hybridobjekte[6] genannt haben. Beispiele für Hybridobjekte sind:

- die „Turingmaschine", die 1936 von Alan Turing als Berechnungsformalismus eingeführt wurde. Sie ist Formalismus - Notation - und sich bewegende Maschine gleichzeitig. Sie beschreibt sowohl den menschlichen als auch den maschinellen Rechner (vgl. Abschnitt 2.2.2);

---

3 Ich werde – im Gegensatz zu vielen anderen Arbeiten, in denen Wissenschafts- und Technikentwicklung als kultureller Prozeß begriffen wird – an Stelle des Begriffs der Deutung zumeist auf den Begriff Sinngeflecht oder Orientierungsrahmen zurückgreifen. Im Begriff der Deutung schwingt mit, daß es sich hier um eine intentionale Handlung handelt, bei der bewußt eine Gegenstand oder Sachverhalt gedeutet wird. Solche „Deutungen" erfolgen jedoch zumeist implizit und werden nicht bewußt von den jeweiligen Akteuren betrieben.

4 Das interdisziplinäre Forschungsprojekt „Sozialgeschichte der Informatik", in dem ich als Wissenschaftliche Mitarbeiterin tätig war, wurde von 1993 bis 1997 an der Technischen Universität Berlin gefördert. In diesem Projekt erarbeiteten Informatiker zusammen mit Geistes- und Sozialwissenschaftlern neue Ansätze der Wissenschafts- und Technikforschung in direktem Zusammenhang mit historischen Studien zu diversen Aspekten der Informatikentwicklung. Die Ergebnisse des Projektes sind in (Siefkes et al. 1998) sowie in (Eulenhöfer et al. 1998) zusammengefaßt.

5 Diese Ergebnisse hat mein Kollege Peter Eulenhöfer im Rahmen seines Dissertationsvorhabens (Eulenhöfer 1998b) und einem Artikel (Eulenhöfer 1998a) ausgearbeitet.

6 „hybrid" stammt aus dem Griechischen und bedeutet „von zweierlei Abkunft, gemischt, zusammengesetzt, zwitterhaft". Der Begriff „Hybridobjekt" soll darauf verweisen, daß hier in der Sicht auf ein Objekt verschiedene Bereiche gemischt werden, die in der Regel als Dichotomien aufgefaßt werden.

- Automaten, wie sie von Neumann 1948 beschrieb. Sie können als Symbolsystem, Maschine oder als natürlicher Organismus vorgestellt werden (vgl. Abschnitt 3.1);
- Programme wie z. B. der „General Problem Solver“ (GPS) von Newell und Simon. Der GPS wurde als psychologische Theorie des menschlichen Denkens – als Beschreibung –, und als maschinelle Simulation menschlichen Problemlösens vorgestellt (zu Newells und Simons Arbeiten vgl. Abschnitt 4.6.1);
- „Objekte“, wie sie im Rahmen der Diskussion um objektorientierte Programmierung aktuell sind. Sie können als Modelle und Softwarekonstrukte – als zeichenhaft und beschreibend – und zudem als handelnde Gegenstände gesehen werden (vgl. Abschnitt 5.2).

Mit dem Begriff „Hybridobjekte“ soll ein Gegenstand oder Gegenstandsbereich charakterisiert werden, so wie er beschrieben und konzipiert wird. „Hybridobjekt“ meint also weniger einen real existierenden Gegenstand als vielmehr eine Konzeption, die jedoch nicht von der Wahrnehmung der realen Gegenstände und den damit verbundenen Handlungsentwürfen getrennt werden kann. „Hybridobjekt“ verweist auf ein Orientierungsmuster, und ich werde häufig auch von „Hybridobjekt-Sichtweise“ reden. Hybridobjekte lassen sich auf einer allgemeineren Ebene wie folgt charakterisieren:

- Hybridobjekte erscheinen einerseits als schriftliche Zeichen, Notationen oder Symbolfolgen. Als solche können sie als unvergängliche, die Gedanken konservierende Repräsentanten aufgefaßt werden. Schrift ist in dieser Sicht „die bleibende, die sich über die Sterblichkeit des Menschen hinwegsetzt und seine Gedanken, auch die Erinnerung an ihn selbst erhält, wenn sein Körper schon längst verwest ist“ (von Braun 1989, S. 83). Für die Mathematik konstituiert diese Konstanz der Schriftzeichen eine unwandelbare Welt jenseits der materiellen Veränderlichkeit (Mehrtens 1990, S. 450):

  > „Die Zeichen x und M repräsentieren für sich nichts außer ihrer dauerhaften Identität. Diese Identität allein ist schon nicht von dieser Welt, deren Dinge höchst wandelbar und vergänglich sind. Namen zu geben, Symbole zu setzen, ist von anderer Art. Es ist der Akt, mit dem Dauer und Stabilität geschaffen wird, weil man das Zeichen wiedererkennen und seinen Kontext erinnern kann. Damit läßt sich über eine stabile Welt reden.“

  Die mathematischen Zeichen repräsentieren nach Mehrtens also nicht etwas Zweites, sondern nur „ihre dauerhafte Identität“. Sie bilden Namen, die man wiedererkennen kann und die deshalb erlauben, über eine stabile Welt zu reden. Sowohl von Braun als auch Mehrtens – auch wenn ihre Gedanken sich unterscheiden – betonen den Aspekt der Konstanz der Schriftzeichen: Es ist ihre Form, die immer wiedererkannt werden kann, und damit eine Eigenschaft, die von der materiellen Beschaffenheit abstrahiert, die den Wert der Zeichen als Repräsentanten ausmacht.
- Nun kann andererseits die materielle Realisierung der Zeichen und Notationen betrachtet werden – so z. B., ob sie durch Tinte auf dem Papier oder durch

elektronisch realisierte Schaltungen im Computer dargestellt werden. In dieser Hinsicht erscheinen die Zeichen als vergängliche Gegenstände. Kennzeichnend für die Hybridobjekt-Sichtweise ist darüber hinaus, daß den – einmal als Repräsentanten, dann als Gegenständen – vorgestellten Notationen auch die Fähigkeit zu agieren zugesprochen wird. Sie scheinen wie eine Maschine operieren zu können, und diese Operationen werden auch als Handlungen begriffen, die sie selbsttätig wie ein Mensch ausführen. Die einerseits konstanten, etwas Zweites oder ihre eigene Indentität repräsentierenden Zeichen werden damit auch als gegenständliche Subjekte der Veränderung gedacht.

In der Hybridobjekt-Sicht werden also Notationen, Zeichen und Beschreibungen auch als materiell und agierend gedacht. Sie erscheinen als agierende Notationen. Zeichen werden damit aus ihrem passiven Status gehoben und als aktive Kräfte mit Auswirkungen auf die materielle Welt vorgestellt. Die Dualismen Gegenstand-Zeichen, Repräsentiertes-Repräsentant und materiell-immateriell verschwimmen. Das hat auch Auswirkungen darauf, wie die Herstellung von Hybridobjekten wahrgenommen wird: Ein Hybridobjekt niederzuschreiben wird nicht nur als Beschreiben, sondern genauso als (Auf)bauen und damit Konstruieren interpretiert – ähnlich wie das in dem anfänglich wiedergegebenen Zitat von Zemanek über die Tätigkeit des Informatikers der Fall ist. Denn die aufgeschriebenen Zeichen fungieren zugleich als Beschreibungen und Gegenstände, die bewegen und agieren können.

Daß Hybridobjekten die Kraft zu agieren zugesprochen wird, verweist auf einen weiteren Dualismus, der in der Hybridobjekt-Sichtweise in Frage gestellt wird: den zwischen natürlich und technisch und spezieller zwischen Mensch und Maschine. In den oben angeführten Beispielen werden natürliche Organismen und besonders Menschen, Maschinen und Symbolsysteme bzw. Notation sehr nah aneinander gedacht – etwas, das sich im Verlauf dieser Arbeit immer wieder zeigen wird. Das Verschwimmen der Differenz materiell-immateriell baut auf dem Verschwimmen der Differenz natürlich-technisch auf oder hängt doch eng mit ihm zusammen. Im Begriff des Programmierens – einer Tätigkeit, die auch als Umgang mit Symbolen verstanden wird – spiegelt sich das wider: Als programmiert gelten nicht nur Maschinen, sondern auch das Gehirn und menschliches Denken sowie die Entstehung von Leben. Die Hybridobjekt-Sichtweise und der Umstand, daß das Programmieren als universellere Tätigkeit erscheint als sie das Steuern einer Maschine darstellt, sind eng miteinander verknüpft.

Daß das Auflösen der Dualismen natürlich-technisch sowie Gegenstand-Zeichen, wie es für die Hybridobjekt-Sichtweise kennzeichnend ist, schon seit dem Beginn der Speicherprogrammierung virulent war, zeigt die Analyse von von Neumanns Arbeiten (vgl. Kapitel 2 und 3). Daß die Hybridobjekt-Sichtweise auch in den biologischen Wissenschaften und besonders der Genetik eine wichtige Rolle spielt, wird in Evelyn

Fox-Kellers Buch „Refiguring Life“ deutlich.[7] Keller zeigt hier Zusammenhänge zwischen der Entwicklung der Genetik und der Computertechnologie und Programmierung auf. Sie kommt dabei auf eine prägnante Formulierung Norbert Wieners[8] in „God and Golem, Inc.“ zu sprechen, welche die Grundidee des Cyberspace bereits 1964 umreißt (Wiener nach Keller 1995, S. 106): „That is an idea with which I have toyed before – that it is conceptually possible for a human being to be send over a telegraph line“. Keller hält dazu fest (Keller 1995, S. 115):

> „Wiener was not simply erasing the distinction between bodies and machines; he was also relying on the conflation between machines and messages that has been endemic in computer discourse since its very beginnings.“

Die Unterscheidung zwischen Körper, Maschine und Nachricht steht also seit den Anfängen der Computerentwicklung und nicht nur im Diskurs der Programmierung zur Disposition. Das Konzept der Hybridobjekte erlaubt damit, auf den ersten Blick neue Diskussionen und Entwicklungen im Bereich der Programmierung wie z. B. die Vorstellung von der „Virtualisierung des Seins“, dem „virtuellen Marktplatz“, der „virtuellen Gemeinschaft“, „Cyberspace“ etc. als Teil einer längeren Kontinuität zu begreifen, und nach den Wurzeln dieser Konzeption in der ersten Hälfte und Mitte dieses Jahrhunderts zu fragen.

Ein Anliegen der vorliegenden Arbeit ist auch, einen Teil der wissenschaftlichen Diskursserien zu rekonstruieren, die konstitutiv für die Programmierung und besonders auch die Hybridobjekt-Sichtweise in diesem Bereich waren. Die dabei zu erzählenden Geschichten werden zeigen, daß nicht nur elektrotechnische und mathematische Diskurse hier eine entscheidende Rolle spielten. Wichtig waren ebenso neurophysiologische und behavioristische Anschauungen. Für die Herausbildung der Programmierung war und ist bis heute ein behavioristischer Handlungsbegriff zentral, nach dem Handeln aus dem Ablauf von Reiz-Reaktionsmechanismen erklärt werden kann. Menschliches Handeln und maschinelle Operation sind vor dem Hintergrund dieses Handlungsbegriffs in gleicher Weise erklärbar und können in gleicher Weise als Symbolsysteme repräsentiert werden. Menschliches Denken und Handeln kann als programmiert gedacht werden.

Ein bestimmter Handlungsbegriff legt eine bestimme Vorstellung davon, wie Handeln erschöpfend beschrieben und erklärt werden kann, und damit bestimmte Methoden und Konzepte nahe. Während ich in dieser Arbeit von einem Handlungsbegriff ausgehe, nach dem Handeln nicht losgelöst von seinem Sinn verstanden werden kann, zeige ich für den Diskurs der Programmierung auf, daß hier von Beginn an ein behavioristischer Handlungsbegriff zentral war, der die Frage nach dem Sinn des

---

7 In Abschnitt 3.2.1 behandele ich Bezüge zwischen Programmierung und Genetik.

8 Auf Wieners Arbeiten werde ich besonders in Abschnitt 3.1.1 zu sprechen kommen. Da Wiener im Bereich der Programmierung kaum aktiv war, gehe ich dabei jedoch nicht zu sehr in die Tiefe.

Handelns unbeantwortet läßt. Mit diesem Handlungsbegriff sind im Rahmen der Programmierung entsprechende methodische Verfahren und Konzepte verbunden (vgl. Abschnitt 5.2). In einer Wissenschaft, deren Herzstück die Programmierung ist – der Informatik –, ist die Frage nach dem Sinn und der konstitutiven Kraft, die Sinnstrukturen in dieser Wissenschaft haben, und damit das methodisches Vorgehen dieser Arbeit eher ungewöhnlich.[9] Insofern hat die vorliegende Arbeit neben einem inhaltlichen auch ein methodisches Anliegen. Auf der inhaltlichen Ebene soll aufgezeigt werden, in welche Sinnzusammenhänge die Programmierung eingebettet wurde, und wie das die technisch-wissenschaftlichen Projekte in diesem Bereich prägte und prägt. Auf der methodischen Ebene soll ein Vorgehen entwickelt und in den einzelnen Textbesprechungen exemplifiziert werden, das erlaubt, technisch-wissenschaftlichen Texten unterliegende Sinnstrukturen herauszuarbeiten. Für den Bereich der Softwareentwicklung integriere ich dieses Vorgehen in Abschnitt 5.3 in einen breiteren Methodenrahmen, der erlaubt den Sinn mitzuberücksichtigen, den die einzelnen beteiligten Gruppen und Personen in eine Systementwicklung hineintragen. Die Arbeit soll damit auch dazu beitragen, einen konstruktiven Umgang mit der Gesellschaftlichkeit und Kulturalität von Technik und Wissenschaft nicht nur im Bereich der Programmierung zu finden. Statt sie als Störfaktoren im Technikentwicklungsprozeß zu begreifen, soll das kreative Potential unterliegender Sinnstrukturen in der Technikentwicklung sicht- und nutzbar gemacht werden.

---

9 Aber nicht einmalig. So werden z. B. in den Arbeiten von Naur und Floyd Fragen des Sinns miteinbezogen (vgl. Abschnitt 5.2).

Kapitel 1: Wissenschafts- und Technikentwicklung als Denk- und Handlungspraxis

Wie die Gestaltungsmöglichkeiten im Bereich der Informationstechnologie eingeschätzt werden, ist eng gebunden an die generelle Sicht auf Naturwissenschaft und Technik, die eine Gesellschaft, bestimmte gesellschaftliche Gruppen und besonders die in der Technik- und Wissenschaftsentwicklung tätigen Personen sich zu eigen machen. Wird Naturwissenschaft als Suchen und Finden immer neuer Gesetzlichkeiten und Wahrheiten, die so in der Welt vorgefunden werden, begriffen, so reduzieren sich die Handlungsmöglichkeiten von Wissenschaftlerinnen und Technikern im wesentlichen darauf, sich an mehr oder weniger zwangsläufig scheinenden Entwicklungen zu beteiligen oder aber individuell davon abzulassen. Wird hingegen Wissenschafts- und Technikentwicklung als sozialer und kultureller Prozeß verstanden, ergeben sich neue Handlungsspielräume. Ähnlich wie in anderen gesellschaftlichen Bereichen geht es unter dieser Prämisse nicht mehr nur darum, wahr und falsch (bzw. funktionierend und nicht funktionierend) gegeneinander abzuwägen, sondern auch darum, die soziale und kulturelle Eingebundenheit der Erkenntnisse zu reflektieren. Die vielen alltäglichen (Denk-)Handlungen, die einer wissenschaftlichen Theorie oder einer Technik vorausgehen, werden nun zum Thema. Eine technische oder wissenschaftliche Entwicklung zu gestalten bedeutet jetzt auch, diese Handlungen zu thematisieren und gerade auf solche Vorannahmen und Selbstverständlichkeiten hin zu befragen, die in der Regel nicht reflektiert werden und gerade deshalb zwanghaft wirken können. Ein Zwang, der eine wissenschaftliche Entwicklung scheinbar vorantreibt, entpuppt sich so unter Umständen als habituell verankerte Handlungspraxis, als Denkgewohnheit, die durchaus – und damit auch die entstehende Theorie oder Technik – veränderbar ist.

Zumindest theoretisch wird heute Wissenschafts- und Technikentwicklung als sozialer und kultureller Prozeß aufgefaßt; in der alltäglichen Handlungspraxis ist diese Einsicht jedoch nur wenig wirksam. Die hier angestellten Überlegungen sollen deshalb zu einem praxisbezogenen Ansatz beitragen, der ermöglicht, konstruktiv mit gesellschaftlich-kulturellen Einflüssen und besonders mit unterliegenden Sinnstrukturen in der Technik- und Wissenschaftsentwicklung umzugehen.

Um diese Arbeit zunächst in einem breiteren methodisch-theoretischen Kontext zu verorten, gebe ich in diesem Kapitel zunächst einen kurzen Überblick über verschiedene Ansätze, Wissenschafts- und Technikentwicklung als gesellschaftlich-kulturelles Unternehmen zu verstehen. Besonders gehe ich dabei auf Fleck und Foucault ein, die beide die Wichtigkeit impliziten Wissens in der Wissenschafts- und Technikentwicklung herausgearbeitet haben, und damit als Wegbereiter der in dieser Arbeit gewählten Sicht auf Wissenschaft und Technik gesehen werden können. Implizites Wissen steht den einzelnen Personen zwar zur Verfügung, kann aber von ihnen nicht

formuliert werden oder erscheint so selbstverständlich, daß es nicht formuliert wird. Mit den Begriffen Denkkollektiv und Denkstil, Diskurs und Orientierungsmuster, die in diesem Kapitel vorgestellt werden, führe ich eine Terminologie ein, mit der implizites Wissen genauer gefaßt werden kann. Vor diesem Hintergrund stelle ich anschließend die Methodik vor, mit deren Hilfe in dieser Arbeit der Entwicklung der Programmierung und von Programmiersprachen unterliegende implizite Wissensbestände herausgearbeitet werden: die Dokumentarische Interpretation. Da es für die Analyse technisch-wissenschaftlicher Texte meines Wissens noch keine angepaßte Methode gibt, die erlaubt, die kulturelle Gewordenheit dieser Dokumente zu analysieren, übertrage ich die Dokumentarische Interpretation aus dem Gebiet der rekonstruktiven Sozialforschung, wo sie primär zur Analyse narrativer Interviews und von Gruppendiskussionen verwendet wird, auf die Interpretation wissenschaftlich-technischer Texte. Das verweist auf die grundlegende These der vorliegenden Arbeit: Manifestationen technisch-wissenschaftlicher Arbeit sind prinzipiell als Produkte von Kultur und Gesellschaft interpretierbar wie andere Manifestationen von Kultur und Gesellschaft – Gespräche, Romane, Theaterstücke oder Philosophische Texte – auch.

1.1 Implizites Wissen in der Wissenschafts- und Technikentwicklung

Der Alltag in den Industriestaaten ist ohne die Produkte von Wissenschaft und Technik und besonders auch die Computertechnologie nicht mehr denkbar. Dennoch wird der Frage, wie die wissenschaftlichen Erkenntnisse und die Technologien zustande kommen, die in unserem Leben so selbstverständlich sind, wenig Beachtung geschenkt. Immer noch bestimmen traditionelle Vorstellungen über Wissenschaft und Technik, die selbst in vielen sich durchaus kritisch verstehenden Beiträgen reproduziert werden, die Diskussion. Der Wirkungszusammenhang von Wissenschaft und Technik auf der einen, und Gesellschaft und Kultur auf der anderen Seite erscheint in dieser Sicht unidirektional: In Wissenschaft und Technik werden neue Erkenntnisse und Produkte entwickelt, die scheinbar unaufhaltsam die Grundlage unseres Zusammenlebens verändern. Den Menschen außerhalb der engeren Wissenschafts- und Technikzirkel kommt dieser Sicht auf die Wissenschafts- und Technikentwicklung zufolge ein passiver Part zu: als Nutzerinnen und Nutzer, als Betroffene, bestenfalls als Nachfragende eines Marktes.

Bei einer solchen Einschätzung des Wirkungszusammenhangs zwischen Technik und Wissenschaft auf der einen, sowie Gesellschaft und Kultur auf der anderen Seite werden in der Regel nur die Folgen von Wissenschaft und Technik auf die Gesellschaft, die Umwelt, die Kultur oder den Frieden debattiert. Lange Zeit galt daher die Technologiefolgenabschätzung als der einzige Zugang zu einer gesellschaftlichen

Auseinandersetzung mit der wissenschaftlichen und technologischen Entwicklung. Seit einigen Jahren wird jedoch immer häufiger Kritik an einer solchen einseitigen Folgeabschätzung laut. „Mit dem Blick auf die Folgen übernähmen wir implizit einen technologischen Determinismus" stellt beispielsweise Rammert (1998, S. 51f.) fest:

> „Sehen wir die Technik jedoch als einen offenen Prozeß der Technisierung, der erst über die Visionen, die in der Gesellschaft kursieren, ausgerichtet wird und in konkreten Projekten von sozialen Akteuren Gestalt annimmt [...], dann wachsen die Chancen, die gesellschaftlichen Herausforderungen für die Technik nicht nur zu formulieren, sondern die technische Entwicklung auch in gewünschte Richtungen zu steuern".

Seit den 70er Jahren mehren sich empirische und theoretische Arbeiten, die die traditionelle Sicht auf die Entwicklung von Wissenschaft und Technik als fragwürdig erscheinen lassen. So wurde beispielsweise von Kuhn (1962) und Feyerabend (1975) am Beispiel der Naturwissenschaften die Vorstellung vom kumulativen Anwachsen des Wissens und von der Neutralität der wissenschaftlichen Bewertungsmaßstäbe in Frage gestellt. Dadurch wurde eine Serie von empirischen Detailstudien über Praktiken in Wissenschaft und Technik angeregt; die Neubewertung der Rolle von Technik und Wissenschaft in der Gesellschaft erwies sich als sinnvoll. Es etablierten sich verschiedene Erklärungsmodelle von wissenschaftlicher und technischer Entwicklung.[10] Zunächst wurde der Stellenwert expliziter Interessen verschiedener Akteursgruppen auf die Wissenschafts- und Technikentwicklung herausgearbeitet – eine Sichtweise, die ich in diesem Abschnitt zunächst kurz vorstelle. Soziale und kulturelle Faktoren erscheinen in dieser Sicht allzu leicht als Einflußfaktoren auf eine, tendenziell weiterhin scheinbar einer eigenen Dynamik folgenden Wissenschafts- und Technikentwicklung. Betrachtet man – wie es das Ziel der vorliegenden Arbeit ist – Wissenschaft- und Technikentwicklung konsequent als sozialen und kulturellen Prozeß, rückt an Stelle außerhalb der Wissenschafts- und Technikentwicklung liegender Faktoren das alltägliche Denken und Handeln der Wissenschaftlerinnen und Techniker selbst in das Zentrum der Betrachtung. Über explizite Interessen hinaus wird damit auch implizites Wissen – nicht reflexiv verfügbare, aber dennoch das Denken und Handeln orientierende Gewohnheiten und Praxen – für die Untersuchung von Technik- und Wissenschaftsentwicklung zentral. Im Rahmen der Wissenschafts- und Technikforschung haben Fleck und Foucault, auf deren Arbeiten ich deshalb in diesem Abschnitt ebenfalls eingehe, die Relevanz impliziten Wissens herausgearbeitet. Um die handlungsleitende Kraft impliziten Wissens begrifflich begrifflich besser fassen zu können, erläutere ich anschließend das Konzept der Orientierungsmuster, so wie es Bohnsack beschreibt.

---

10 Einen relativ ausführlichen Überblick gibt z. B. Heintz (1993, Kap. 3.1). In der recht amüsanten Form eines detektivischen Dialogs stellen Handel und Hess (1998) sozialgeschichtliche Ansätze zur Wissenschafts- und Technikforschung vor.

Eine Möglichkeit, sich der Sozialität von Wissenschaft und Technik zu nähern, besteht darin, Einflüsse, die in Form finanzieller Zuwendungen oder politischer Entscheidungsmacht einen Theorie- oder Technikentwicklungsprozeß kanalisieren, zu untersuchen. Dabei geht es darum, die Überformung wissenschaftlicher und technischer Ergebnisse durch Interessenkonstellationen, die sich im Zusammenwirken der beteiligten Gruppen und Institutionen ergeben, herauszuarbeiten. Für die Informatikentwicklung wurde in diesem Sinne beispielsweise untersucht, wie militärische Interessen die Computerentwicklung (vgl. Lindner et al. 1988) beeinflußt haben und wie die Technologie numerisch gesteuerter Werkzeugmaschinen und weite Teile der Softwareengineering-Debatte durch Interessengegensätze zwischen Management und Maschinen- bzw. Programmierexperten geprägt wurden (vgl. Noble 1986b, Kraft 1977).

Gesellschaft, Wirtschaft und Politik erscheinen in diesem Modell als Einflußfaktoren auf die Technik- und Wissenschaftsentwicklung. Sie bestimmen im wesentlichen, welche technisch oder wissenschaftlich schon gegebenen Möglichkeiten ausgewählt werden, erscheinen also als Machtfaktoren: Gefragt wird z. B. nach den Kräftekonstellationen der beteiligten Akteure, danach, welche technisch-wissenschaftlichen Alternativen sich durchsetzten, und durch welche Interessen die so sondierten informatischen Produkte schließlich geprägt sind. Wie es hingegen dazu kam, daß nach der Finanzierung eines Vorhabens eine Technik oder Theorie gerade in dieser Weise entwickelt wurde, warum bestimmte Ideen gerade diese und keine andere Form hatten, ist so nicht erschöpfend zu beantworten. Der Blick auf die Technik- und Wissenschaftsentwicklung ist tendenziell ein Blick von außen, und es besteht die Gefahr, daß diese im Kern weiterhin als Bereich vorgestellt wird, der zwar mit Gesellschaft und Kultur verknüpft, aber doch von diesen grundverschieden ist.

Häufig liegt Untersuchungen dieser Art eine Theorie darüber zugrunde, wie gesellschaftliche und ökonomische Macht in einer Gesellschaft verteilt sind, und wie sich diese Machtverteilung auf die wissenschaftliche und technische Entwicklung auswirkt. Ausgegangen wird von einer Reihe von Einflußfaktoren, die als wichtig bei der Betrachtung technisch-wissenschaftlicher Entwicklungen angesehen werden; eine solche Sicht betont die Einheitlichkeit oder Strukturähnlichkeit von Entwicklungen in Wissenschaft und Technik allgemein.[11] Da die Fragestellungen von vornherein festliegen, können neue Fragen, die sich aus der Spezifik und Besonderheit einzelner Geneseprozesse ergeben, nur schwer formuliert werden. Besonderheiten einzelner Entwicklung geraten dabei leicht aus dem Blick.

---

11 Ein für das hier geschilderte Herangehen beispielhaftes Modell der Technikgeschichtsschreibung ist z. B. das Akteurs-Struktur-Modell, das in (König 1993) dargestellt ist.

Gibt man die Dichotomie von Wissenschaft und Technik auf der einen sowie Gesellschaft und Kultur auf der anderen Seite prinzipiell auf und versteht Wissenschafts- und Technikentwicklung als einen sozialen und kulturellen Prozeß, so stellt sich die Arbeit, das Denken und Handeln der Wissenschaftler und Technikerinnen selbst als durchdrungen von Wertungen, Gewohnheiten und Sinngebungen dar. Wissenschaftlerinnen und Techniker greifen nach dieser Vorstellung bei ihrer Arbeit – zumeist ohne dies zu bemerken – auf Denk- und Handlungsmuster aus dem Alltag, dem sozialen Umfeld, der Kultur zurück, so daß die wissenschaftlichen und technischen Ergebnisse nicht zuletzt als Ausdruck von Deutungen, kulturellen Werten, Traditionen und Handlungspraxen erscheinen. Implizites Wissen wird nun als relevanter Faktor in der Wissenschafts- und Technikgestaltung begriffen; die explizierten Interessen und Absichten einzelner Akteure oder Akteursgruppen können selbst Ausdruck solcher impliziten Orientierungen sein oder zumindest mit diesen harmonieren. Explizite und implizite Orientierungen können aber auch in einem Spannungsverhältnis stehen oder sogar gegensätzlich sein. Wissenschaftliche und technische Entscheidungen und Konzepte fußen demnach nicht auf einer wissenschafts- oder technikinternen Rationalität; diese Rationalität wird selbst als Ausdruck von Auffassungen, Praktiken und Wertungen begriffen. Technisch-wissenschaftliche Experten verfügen damit nicht über ein Wissen, das jenseits gesellschaftlicher und kultureller Werte als wahr gelten kann, sondern ihr Tun, ihre Ergebnisse und Produkte sind durch und durch kulturell und sozial.

Ein frühes Beispiel für eine solche Perspektive sind die wissenschaftsphilosophischen Arbeiten des polnischen Mediziners Ludwik Fleck. Fleck versuchte, mit Hilfe der Begriffe „Denkstil“ und „Denkkollektiv“, auf die ich in den folgenden Kapiteln zurückgreifen werde, die Wirkungsweise impliziten Wissens in der Wissenschafts- und Technikentwicklung genauer zu fassen. Schon 1935 schrieb er das Buch „Die Entstehung und Entwicklung einer wissenschaftlichen Tatsache“. Dieses wendete sich gegen die Wissenschaftsauffassung des Wiener Kreises, wie sie in Poppers „Die Logik der Forschung“ formuliert ist. Fleck prägte hier den Begriff „Denkkollektiv“. Deutlich wird anhand dieses bei Fleck für die Erklärung von Wissenschaftsentwicklung zentralen Begriffs, was in der Einleitung des Buches festgehalten wird (Schäfer, Schnelle 1994, S. VIII): „Wissenschaft war für ihn kein formales Konstrukt, sondern wesentlich eine Tätigkeit, veranstaltet von Forschergemeinschaften.“ Den Begriff „Denkkollektiv“ faßte Fleck wie folgt (Fleck 1935, S. 54f.):

> „Definieren wir ‚Denkkollektiv‘ als Gemeinschaft der Menschen, die im Gedankenaustausch oder in gedanklicher Wechselwirkung stehen, so besitzen wir in ihm den Träger geschichtlicher Entwicklung eines Denkgebiets, eines bestimmten Wissensbestandes und Kulturstandes, also eines besonderen Denkstils.“

Ein Denkkollektiv zeichnet sich nach Fleck dadurch aus, daß es einen gemeinsamen „Denkstil“ pflegt. Dieser Denkstil legt fest, was in einer Gruppe als wahr bzw. falsch

begriffen wird, ohne daß Wahrheit lediglich als Konvention oder Übereinkunft erscheint (Fleck 1935, S. 130f.):

> „Wie können also *Denkstil als gerichtetes Wahrnehmen, mit entsprechendem gedanklichen und sachlichen Verarbeiten des Wahrgenommenen, definieren.* Ihn charakterisieren gemeinsame Merkmale und Probleme, die ein Denkkollektiv interessieren; die Urteile, die es als evident betrachtet; der Methoden, die es als Erkenntnismittel anwendet. [...] Gehören A und B demselben Denkkollektive an, dann ist der [selbe] Gedanke entweder für beide wahr oder falsch. [...] Wahrheit [ist] nicht Konvention, *sondern im historischen Längsschnitt: denkgeschichtliches Ereignis, in momentanem Zusammenhange: stilgemäßer Denkzwang.*"

Denkstile sind nach Fleck historisch gewachsen, gesellschaftlich bedingt und miteinander durch ihre menschlichen Träger, die sie weiterentwickeln, verwoben. Das Wissensgebäude eines Denkkollektivs, das was hier als wahr befunden wird, ist von unterliegenden Vorstellungen abhängig, die ihrerseits an eine Gesellschaft, Kultur und geschichtliche Situation gebunden sind.

Denkkollektive bestehen nach Fleck zunächst aus dem „esoterischen Kreis" der „Eingeweihten", also der Wissenschaftler und Wissenschaftlerinnen, die die aktuelle Forschungsspitze verkörpern. Ebenso gehört zu einem Denkkollektiv aber auch ein „exoterischer Kreis", der die Ideen des esoterischen Kreises aufnimmt und popularisiert – „gebildete Laien" wie z. B. Schullehrer oder Wissenschaftsjournalisten. Wichtig ist, daß nicht nur der exoterische von dem esoterischen Kreis beeinflußt wird, sondern auch umgekehrt (Fleck 1935, S. 139):

> „Doch auch diese Eingeweihten sind keineswegs unabhängig: sie sind mehr oder weniger – bewußt oder unbewußt – von der ‚öffentlichen Meinung', d. h. der Meinung des exoterischen Kreises abhängig. Auf diese Weise entsteht im allgemeinen die innere Geschlossenheit des Denkstils und dessen Beharrungstendenz."

Während der exoterische Kreis das Wissen des esoterischen Kreises verallgemeinert und für Laien zugänglich macht, tritt dem esoterischen Kreis das popularisierte Wissen als öffentliche Meinung entgegen und dient als Ressource der Legitimation. Wissenschaft ist also nicht unabhängig von populären Vorstellungen und Ansichten der entsprechenden Zeit zu verstehen. Allgemein sind Wissenschaft und Technik nicht, so Fleck, losgelöst von Gefühlslagen zu sehen, die in einer Gesellschaft zu einer bestimmten Zeit geteilt werden (Fleck 1935, S. 67 f.):

> „Es gibt keine Gefühlsfreiheit an sich oder reine Verstandesmäßigkeit an sich – wie wären sie nur festzustellen? Es gibt nur Gefühlsübereinstimmung und Gefühlsdifferenz, und die gleichmäßige Gefühlsübereinstimmung einer Gesellschaft heißt in ihrem Bereiche Gefühlsfreiheit. Sie ermöglicht ein ohne große Deformierung mitteilbares, d. h. ein formales, schematisches, in Worten und Sätzen erfaßbares Denken, dem eine Gewalt, unabhängige Existenzen zu statuieren, eben gefühlsmäßig zuerkannt wird. Solches Denken heißt dann verstandesmäßig."

Die Trennung zwischen Verstand und Gefühl, zwischen einem rationalen und einem emotionalen Bereich ist demnach obsolet. Was als rational angesehen wird, beginnt nicht etwa dort, wo Gefühle ausgeschaltet sind, sondern dort, wo sie in einer Gesellschaft allgemein geteilt und deshalb unsichtbar werden.

Flecks Werk selbst war vielleicht der beste Beleg für seine Wissenschaftsauffassung (Schäfer, Schnelle 1994, S. VIIIf.):

> „Die im wahrsten Sinne des Wortes externen Bedingungen der Wissenschaft, die Fleck in seinem Buch erörtert hatte, ließen eine Rezeption kaum zu. Der polnische Jude Fleck konnte im Deutschland der Nazis kein Interesse finden."

Fleck wurde nach der deutschen Besetzung 1941 zunächst in das Ghetto von Lwów und anschließend in die Konzentrationslager Auschwitz und Buchenwald, die er und auch seine Frau und sein Sohn überlebten, deportiert. Nach dem Krieg widmete er sich nur noch der medizinischen Arbeit, so daß seine Ideen nur vermittelt über Kuhns „Die Struktur der wissenschaftlichen Revolutionen" in den wissenschaftstheoretischen Diskurs Eingang fanden.[12] Erst in letzter Zeit wird Flecks Werk verstärkt rezipiert.

Ein weiteres Beispiel für die Sicht auf Wissenschafts- und Technikentwicklung als sozialer und kultureller Prozeß sind die Arbeiten Michel Foucaults. Wie bei Fleck wird auch bei ihm die Wichtigkeit impliziten Wissens für das wissenschaftliche und technische Denken und Handeln deutlich. Fast vierzig Jahre nach dem Erscheinen von Flecks „Entstehung und Entwicklung einer wissenschaftlichen Tatsache" formulierte Foucault, zu dieser Zeit Inhaber des Lehrstuhls für die Geschichte der Denksysteme am „Collège de France" in Paris (Foucault 1974, S. 11):

> „Was ich jedoch erreichen wollte, war, ein *positives Unbewußtes* des Wissens zu enthüllen: eine Ebene, die dem Bewußtsein des Wissenschaftlers entgleitet und dennoch Teil des wissenschaftlichen Diskurses ist"

Und einige Seiten später (Foucault 1974, S. 15):

> „Ich wollte gerne wissen, ob die Individuen, die verantwortlich für den wissenschaftlichen Diskurs sind, nicht in ihrer Situation, ihrer Funktion, ihren perzeptiven Fähigkeiten und in ihren praktischen Möglichkeiten von Bedingungen bestimmt werden, von denen sie beherrscht und überwältigt werden."

Foucault sprach auch von einem „historischen Apriori", das

> „in einer bestimmten Epoche in der Erfahrung ein mögliches Wissensfeld abtrennt, die Seinsweise der Gegenstände, die darin erscheinen, definiert, den alltäglichen Blick mit theoretischen Kräften ausstattet und die Bedingungen definiert, in denen man eine Rede über die Dinge halten kann, die als wahr anerkannt wird." (Foucault 1974, S. 204)

Die Entstehung und Gestalt einer wissenschaftlichen Theorie oder Technik wird nach Foucault getragen durch ein Konglomerat von unproblematisierten Denkweisen: das Selbst- und Menschenbild der Akteure, die Begrenzungen der jeweiligen wissenschaftlichen Disziplin, den Zusammenhang der Akteure mit gesellschaftlichen Mächten und den Einfluß mächtiger Diskurse der Zeit. Wie für Fleck ist für Foucault das wissenschaftliche Denken und Handeln durch eine Ebene des Wissens strukturiert,

---

12 In dem Vorwort zu „The Structure of Scientific Revolutions" bemerkte Kuhn, daß Flecks Buch viele seiner Ideen vorweggenommen hätte. Allerdings sah Fleck anders als Kuhn Veränderungen in der Wissenschaft nicht als Revolutionen, sondern als Veränderungen des Denkstils, die passieren, ohne daß dies den beteiligten Wissenschaftlern direkt bewußt ist.

die historisch gewachsen ist und dem Bewußtsein der Einzelnen „entgleitet". Um diese Ebene begrifflich zu fassen, verwendete Foucault den Begriff „Diskurs", auf den ich besonders im zweiten Kapitel zurückgreifen werde. Diskurse drücken sich nach ihm zum einen in der sprachlichen und schriftlichen Äußerung aus, liegen aber auch Handlungspraktiken sowie dem Aufbau von Artefakten zugrunde. So kann, um ein einfaches Beispiel zu geben, ein Diskurs zum Wesen des Kindes sich in Büchern, Praktiken der Erziehung und Artefakten wie Schulmöbeln, die z. B. zu geradem, unbeweglichem Sitzen zwingen, ausdrücken. Der Diskursbegriff ermöglicht damit, dem Wirken impliziten Wissens auf wissenschaftliche Texte wie technische Artefakte gleichermaßen nachzuspüren.

Während wissenschaftliches Denken und Handeln nach Foucault auf der einen Seite durch Prozesse der „Verknappung, aber auch Umgruppierung und Vereinheitlichung der Diskurse" (Foucault 1991, S. 41) kontrolliert und beschränkt wird, wird es auf der anderen Seite durch die Formierung von Diskursen erst ermöglicht, die gegen und durch obige Kontrollprozeduren entstehen. Denn Diskurse haben, so Foucault, nicht nur die Kraft, „die sich der Verneinung entgegensetzt, sondern die Kraft, Gegenstandsbereiche zu konstituieren, hinsichtlich derer wahre oder falsche Sätze behauptet oder verneint werden können." (Foucault 1991, S. 44) Ähnlich wie Flecks Denkstile bilden Diskurse den Hintergrund einer Übereinkunft darüber, was als wahr, gültig oder diskutierbar angesehen wird. Sie schaffen damit einen Gegenstandsbereich, auf dessen Grundlage wissenschaftliches Denken und Handeln erst möglich wird.

Technisch-wissenschaftliche Geneseprozesse erscheinen bei Foucault nicht als Produkt, sondern als Teil gesellschaftlicher Praktiken und Diskurse. Wie bei Fleck ist auch bei Foucault eine strikte Trennung inner- außerwissenschaftlich nicht möglich; während nach Fleck Verstand und Gefühl eng zusammenhängen, erscheinen bei Foucault der Wissenschafts- und Technikbetrieb sowie das wissenschaftlich-technische Denken als Ausdrucksformen der jeweiligen Zeit und sind demnach selbst durch und durch sozial. Gegenüber dem Interessenmodell verschiebt sich bei Foucault der Machtbegriff: Macht meint weniger die Herrschaft der, salopp gesagt, tendenziell „Bösen" über die Masse der „Guten". Statt dessen geht es um „Mikropraktiken der Macht": Nach dieser Auffassung kann Macht auch produktiven Charakter haben, ist im Alltäglichen wirksam und wird von allen Angehörigen einer Gesellschaft – und natürlich auch von Wissenschaftlern und Technikerinnen – häufig unbewußt immer wieder reproduziert.

Wie Fleck beschreibt auch Foucault eine Ebene des Wissens, die sich der expliziten Beschreibung durch die beteiligten Wissenschaftler entzieht. Was in der Wissenschaft in diesem Zeitraum als wahr, was als falsch und was als überhaupt diskutierbar angesehen wird, wird auf dieser Ebene so entscheidend vorstrukturiert, daß davon die Wissenschafts- und Technikentwicklung grundlegend geprägt wird. Wahrheit ist also

nicht absolut, aber auch nicht Konvention; sie wird gesehen als Ereignis, das gesellschaftlich, kulturell und historisch verwurzelt ist und nicht nach Belieben verändert werden kann. Auf wissenschaftliches Denken und Handeln wirkt die Ebene des impliziten Wissens einerseits beschränkend, andererseits ist sie Voraussetzung und Bedingung dafür, daß überhaupt wissenschaftlich gedacht werden kann; denn Kriterien für die Gültigkeit von Aussagen und Einsichten sind hier verankert.

Nun wird in der Regel die soziale Gebundenheit des wissenschaftlichen – und besonders des naturwissenschaftlichen – Wissens als Fehlerquelle betrachtet (Knorr-Cetina nach Bohnsack 1999): „Es wird unterstellt, daß soziale Einflüsse wissenschaftliche Verfahren derart ‚kontaminieren', daß sie zu inkorrekten Ergebnissen führen." Der Soziologe Karl Mannheim, ein Zeitgenosse Flecks, betonte demgegenüber die „Seinsverbundenheit des Denkens" und die

> „ ‚Verwurzelung des Denkens im sozialen Raum', die nicht allein und nicht primär als eine ‚Fehlerquelle' betrachtet werden dürfen, sondern zugleich auch als ‚größere Chance für die zugreifende Kraft dieser Denkweise in bestimmten Seinsregionen' " (Mannheim nach Bohnsack 1999).

Die von Foucault als positives Unbewußtes bezeichnete, und von Fleck als gleichmäßige Gefühlsübereinstimmung einer Gesellschaft charakterisierte, dem wissenschaftlichen Denken und Handeln unterliegende Ebene des impliziten Wissens kann und sollte also auch positiv verstanden werden – als dem Denken einen Standort und eine Perspektive verleihend und damit als Quelle von Kreativität, die produktiv in der wissenschaftlichen und auch technischen Arbeit umgesetzt werden kann (Bohnsack 1999):

> „Aus der Einbindung der Forscherinnen in eine wissenschaftliche wie außerwissenschaftliche Handlungspraxis resultiert ein Erfahrungswissen, welches nur teilweise explizierbar ist, somit wesentlich implizit oder stillschweigend bleibt. Dieses Erfahrungswissen ist unabdingbare Voraussetzung für Erkenntnis und die Kreativität von Theoriebildung."

Um die produktive Kraft impliziten Wissens besser begrifflich fassen zu können, werde ich in dieser Arbeit auf das aus der rekonstruktiven Sozialforschung stammende Konzept der Orientierungsmuster rekurrieren. Der Begriff „Orientierungsmuster" erlaubt, implizites Wissen, wie es von Fleck und Foucault als Vorbedingung und Rahmen wissenschaftlichen Denkens und Handelns beschrieben wird, als handlungsleitenden Faktor in Wissenschaft und Technik zu berücksichtigen. Ich werde dieses Konzept nun genauer vorstellen.

Bohnsack faßt unter dem Begriff „Orientierungsmuster" zwei Formen handlungsleitender Motive zusammen: zum einen solche Motive, die das Handeln im Sinne eines „Orientierungsschemas", d. h. eines übergreifenden, die Zukunft vorwegnehmenden Entwurfs strukturieren. Ein Orientierungsschema wird z. B. in folgendem Satz deutlich: „Ich studiere Informatik, weil ich in der Softwareproduktion arbeiten will." Diese Aussage „basiert auf einem Handlungs-*Entwurf*, genauer: auf Handlungsvollzügen, die

der einzelne ‚modo futuri exacti', d. h. als in der Zukunft bereits ausgeführt antizipiert" (Bohnsack 1998, S. 106). Eine Analyse solchen um-zu-motivierten – d. h. auf die Erfüllung einer Zukunftsvision ausgerichteten – Handelns hat eher rationalistischen Charakter und erfaßt vor allem das zweckrationale Handeln, „welches an normierten oder institutionalisierten Erwartungsmustern und Wissensbeständen orientiert ist" (Bohnsack 1998, S. 108). In bezug auf technisch-wissenschaftliche Texte beantworten Orientierungsschemata in der Regel die Frage, in welcher explizit benannten Absicht eine bestimmte Theorie oder Technik entwickelt wird. Wenn z. B. der Autor eines Textes feststellt, er habe Rechenmaschinen entwickelt, um die Geschwindigkeit des Rechnens zu erhöhen, liegt hier ein Orientierungssschema vor. Es hat rationalistischen Charakter und erfüllt die häufig an Technikentwickler gestellte Anforderung – das normierte Erwartungsmuster –, durch den Technikeinsatz einen Arbeitsablauf zu beschleunigen und dadurch Kosten zu verringern.

Die zweite Form handlungsleitender Motive, die unter den Begriff Orientierungsmuster subsumiert werden, nennt Bohnsack den „Orientierungsrahmen". Dieser erlaubt, ein Orientierungsschema auf seine Verankerung in der Sozialisationsgeschichte und der Handlungspraxis zu befragen. Der Orientierungsrahmen verweist auf „den Rückbezug des Handelns auf ein vergangenes Erlebnis" (Bohnsack 1998, S. 106), wie das z. B. in dem Satz „Ich studiere Informatik, weil ich schon als Kind von der Technik fasziniert war" zum Ausdruck kommt. Für eine Interpretation technisch-wissenschaftlicher Texte, die auf das positive Unbewußte (Foucault) oder den Denkstil (Fleck) abzielt, steht die Analyse des Orientierungsrahmens im Vordergrund. Ziel dieser Analyse ist zu erarbeiten, welche Erfahrungsräume und welcher Habitus den Hintergrund eines Orientierungsschemas bilden. Bezogen auf obiges Beispiel heißt eine Analyse des Orientierungsrahmens also, zu fragen, vor welchem erlebnismäßigen Hintergrund das Um-zu-Motiv „die Geschwindigkeit des Rechnens erhöhen" für den Autor Sinn ergibt und ein wünschenswertes Ziel darstellt. Angestrebt wird damit ein erklärendes oder motivationsmäßiges Verstehen, was nach Weber bedeutet: die „Erfassung des Sinnzusammenhangs, in den seinem subjektiv gemeinten Sinn nach, ein aktuell verständliches Handeln hineingehört" (Weber nach Bohnsack 1998, S. 106). Im Unterschied zur behavioristischen Auffassung, auf die ich in den folgenden Kapiteln noch genauer zu sprechen komme, ist Handeln in dieser Sicht nicht losgelöst von den Sinnzusammenhängen zu verstehen, in die es eingebettet ist.

Die Relevanz von Orientierungsmustern besteht darin, daß mit diesem Konzept erfaßt werden kann, wie einzelne Individuen, aber auch Gruppen und größere soziale Einheiten Ereignisse sinnhaft wahrnehmen, strukturieren oder auch herbeiführen. Zwar werden zunächst individuelle (Denk-)Handlungen bzw. deren Produkte – in meinem Fall vorwiegend technisch-wissenschaftliche Texte – untersucht, diese aber hängen ihrerseits von der jeweiligen gesellschaftlich-kulturellen Zugehörigkeit der Akteure ab.

Denn der Orientierungsrahmen – die Erfahrungsräume und der entsprechende Habitus – einer Person entwickelt sich vor dem Hintergrund der Sozialisationsgeschichte, die mit anderen geteilt wird (Bohnsack 1998, S. 111f.):

> „Biographische bzw. sozialisationsgeschichtliche Hintergründe prägen einen spezifischen Habitus in vielerlei Hinsicht: in milieu- als auch in generations- und in geschlechtsspezifischer etc. Zugleich konstituieren sozialisationsgeschichtliche Gemeinsamkeiten in diesen verschiedenen Hinsichten entsprechende unterschiedliche konjunktive Erfahrungen und Erfahrungsräume unter denjenigen, denen eine entsprechende habituelle Handlungspraxis, ein Habitus gemeinsam ist. Dieser bedingt einerseits die Wahl, genauer: die Selektivität von Um-zu-Motiven [...]. Andererseits gelangt der Habitus aber in der handlungspraktischen Auseinandersetzung mit den institutionalisierten Um-zu-Motiven und zweckrationalen Ablaufmustern von Institutionen und Rollen erst richtig zur Entfaltung."

Daß ein Autor eines technisch-wissenschaftlichen Textes das Um-zu-Motiv „eine Rechenmaschine bauen, um die Geschwindigkeit des Rechnens zu erhöhen" überhaupt benennt, ist demnach vor dem Hintergrund seiner Erfahrungsräume zu verstehen, die er mit anderen Angehörigen besonders der gleichen Generation, des gleichen Geschlechts und Milieus – Bohnsack (1996) unterscheidet hier Bildungs- und sozialräumliche Milieus – teilt. Die Orientierungsmuster, die einem technisch-wissenschaftlichen Text unterliegen, sagen demnach nicht nur etwas über den oder die Autoren aus. Sie sind auch Ausdruck eines wissenschaftlichen Habitus sowie von Denkgewohnheiten, die zu der entsprechenden Zeit breiter geteilt werden. Besonders wenn der Text breit rezipiert wurde bzw. wird, lassen die unterliegenden Orientierungsmuster Rückschlüsse auf die rezipierende wissenschaftliche Community – in Flecks Terminologie: das Denkkollektiv und dessen charakteristischen Denkstil – zu.

Orientierungsmuster gewinnen ihre handlungsleitende Kraft nicht zuletzt dadurch, daß unerledigte Konflikte in dem durch sie strukturierten Handeln implizit bearbeitet werden. Für in der Wissenschafts- und Technikentwicklung relevante Orientierungsmuster kann sich damit die Frage anschließen, welche Konflikte mit ihrer Hilfe implizit bearbeitet werden. Eine Analyse von in einer Technik- und Wissenschaftsentwicklung zentralen Orientierungsmustern ist damit auch ein Beitrag zu der Beantwortung der Frage, was an dieser Entwicklung fasziniert. Denn (Klaus Heinrich nach Trettin S. 31): „In dem, was fasziniert durch die reale Geschichte hindurch sind unerledigte Konflikte, nicht ausgetragene Spannungen, ist das nicht gelöste Problem jeweils präsent." Die Erarbeitung von Orientierungsmustern kann so dazu beitragen zu verstehen, was eine technische oder wissenschaftliche Entwicklung faszinierend und damit zu mehr macht als einer wissenschaftlichen oder technischen Problemlösung.

Das Programmieren ist längst nicht mehr bloß eine Tätigkeit an der Rechenmaschine: es wird auch als Selbsttechnik gehandelt („ich muß mich mal eben umprogrammieren") und hat sich zu einer Sicht auf die Entstehung des Lebens allgemein gemausert. Die Faszination, die für Viele von der Programmierung ausgeht,

verweist vielleicht auch darauf, daß im Diskurs der Programmierung ein Konflikt mitbearbeitet wird, der in unserer Gesellschaft und Kultur einen zentralen Platz einnimmt. Der Begriff der Programmierung erlaubt – wie ich in den folgenden Kapiteln noch zeigen werde – althergebrachte Dichotomien implizit aufzulösen: die zwischen natürlich und technisch und die zwischen materiell und immateriell. Das Programmieren erscheint vor diesem Hintergrund als eine Art universelle Technik, die in verschiedensten Bereichen der Wirklichkeit einsatzfähig ist und erklärende Kraft beanspruchen kann. Das nicht gelöste Problem – wie denn natürlich und technisch bzw. materiell und immateriell als ein Bereich gedacht werden können – ist in dem Begriff der Programmierung bzw. des Programms jedoch präsent. Die faszinative Kraft, die von der Programmierung ausgeht, erklärt sich auch aus den nicht ausgetragenen Spannungen zwischen natürlicher und technischer sowie materieller und immaterieller Welt, die diesem Begriff, so wie wir ihn denken, inhärent sind.

1.2 Dokumentarische Interpretation technischer Texte

Geht man davon aus, daß Wissenschafts- und Technikentwicklung ein kultureller, an historisch gewordene Denkgewohnheiten und Praktiken gebundener Prozeß ist, so muß die Untersuchung von Wissenschafts- und Technikgeneseprozessen auf einer empirischen Grundlage erfolgen.[13] Denn bei einer solchen Sicht auf Wissenschaft und Technik ist nicht möglich, davon auszugehen, daß Wissenschaftler und Technikerinnen bei ihrer Arbeit einer bestimmten, allgemein verbindlichen Logik oder Vorgehensweise folgen. Um über ihre Art, zu arbeiten und zu denken, etwas aussagen zu können, muß demzufolge von den tatsächlichen Geschehnissen, von den verfügbaren Dokumenten oder Artefakten ausgegangen werden. Bei Fleck wie auch bei Foucault bildeten entsprechend bestimmte wissenschaftliche Ereignisse den Ausgangspunkt ihrer Überlegungen. So war für Foucault zentral,

> „die verschiedenen, verschränkten, oft divergierenden, aber nicht autonomen Diskursserien zu erstellen, die den ‚Ort' eines Ereignisses, den Spielraum seiner Zufälligkeit, die Bedingungen seines Auftretens umschreiben lassen" (Foucault 1991, S. 30).

Es ging ihm um die Rekonstruktion der Bedingungen, die vorhanden sein mußten, damit das je zeitgenössisch Denkbare, überliefert in Gestalt der Dokumente und Quellen, überhaupt ans Licht der Welt treten konnte. Geschichte so verstanden bedeutet eine „Produktionsgeschichte", die die zugänglichen Quellen zunächst als für sich stehende „Monumente" betrachtet und nicht als Ausdruck eines Stifters oder eines überindividuellen Geistes (z. B. des technischen Fortschritts) (vgl. dazu Sonntag

---

13 Ansonsten, so zeigt Elias (1985) anhand von Poppers „Logik der Forschung", wird ein unüberprüfbares Idealbild der Wissenschaft konstruiert.

1992). Die relevanten Fragen werden von hier aus erarbeitet und die Diskursserien, Denkweisen und sozialen Praktiken rekonstruiert, die das Ereignis in dieser Form ermöglichten.

In den folgenden Kapiteln werde ich entsprechend dieser Sichtweise bestimmte Ereignisse aus der Entwicklung der Programmierung und von Programmiersprachen zum Ausgangspunkt meiner Untersuchung machen. Dabei analysiere ich vorwiegend im Rahmen der verschiedenen Entwicklungen entstandene Texte, um einen Zugang zu dem Orientierungsrahmen zu finden, vor dessen Hintergrund diese Entwicklungen verstanden werden müssen. Oft authentischer als z. B. Interviews, in denen gerade Wissenschaftler und Technikerinnen dazu neigen, ihr eigenes Verhalten zu analysieren und im Nachhinein zu erklären, können die von ihnen verfaßten Texte Aufschluß über Selbstverständlichkeiten und Denkgewohnheiten geben. Für meine Analysen greife ich auf das Verfahren der Dokumentarischen Interpretation zurück, mit dessen Hilfe Texten unterliegende Orientierungsmuster herausgearbeitet werden können. In diesem Abschnitt stelle ich dieses Verfahren kurz vor und übertrage es auf den speziellen Anwendungsbereich der technisch-wissenschaftlichen Texte. Zumal ich zugunsten einer besser lesbaren und ergebnisorientierten Darstellung in den folgenden Kapiteln auf eine genaue Beschreibung der verwendeten interpretatorischen Mittel weitgehend verzichte, stelle ich dabei die diese Arbeit strukturierenden Fragen und Interpretationsansätze überblicksartig dar. Insbesondere erörtere ich, wie technisch-wissenschaftliche Texte in Hinblick auf ihren Stil und ihre Metaphorik befragt werden können, und welche Rückschlüsse auf den Orientierungsrahmen von hier aus möglich sind. Da ein methodischer Leitfaden für die Interpretation technisch-wissenschaftlicher Texte weitgehend aussteht, versteht sich die Auflistung der verwendeten interpretatorischen Elemente als eine Art Werkzeugsammlung, die der Vervollständigung und Ausarbeitung harrt, aber für ähnlich gelagerte Untersuchungen eine Hilfestellung bieten kann.

Die Dokumentarische Interpretation dient dazu, Zugang zu den Sinnzusammenhängen zu finden, in die andere Menschen ihre Handlungen und sprachlichen Äußerungen einbetten. In Bereichen, in denen Menschen Erfahrungsräume auf Grund sozialisationsgeschichtlicher Gemeinsamkeiten teilen – in denen sie „in übereinstimmender Weise über gemeinsame stillschweigende, habitualisierte und inkorporierte Wissensbestände verfügen“ – ist ein mehr oder weniger selbstverständliches „unmittelbares Verstehen“ zwischen ihnen möglich; eine Explikation dieser Sinnzusammenhänge ist auf Grund der existentiellen Verbundenheit für diese Personen selbst jedoch problematisch (vgl. Bohnsack 1999). In anderen Bereichen muß man versuchen, auf anderem Wege Zugang zu dem Orientierungsrahmen, den Denkgewohnheiten und dem Habitus zu finden. Die dokumentarische Interpretation bietet hier eine Möglichkeit. Sie stammt aus der rekonstruktiven Sozialforschung und wird hier in der Regel zur

Analyse von Transkripten lebensgeschichtlicher Interviews oder von Gruppendiskussionen eingesetzt. Ausführlich beschrieben ist sie in Hinblick auf diesen Anwendungsbereich in (Bohnsack 1993).

Im Gegensatz zu Interviewtranskripten sind technisch-wissenschaftliche Texte von Spuren der Sozialisationsgeschichte ihrer Autoren weitgehend bereinigt. Trotzdem können wesentliche Elemente der dokumentarischen Interpretation auch für die Interpretation technisch-wissenschaftlicher Texte, die auf das positive Unbewußte oder den Denkstil zielt, verwendet werden. Insofern als es nicht um die Frage der spezifischen, sozialisationsgeschichtlichen Verankerung und Entstehung des Orientierungsrahmens geht, sondern darum, welche Denkgewohnheiten sich in einem Text ausdrücken und dokumentieren, ist diese Methode auf die Interpretation technisch-wissenschaftlicher Texte übertragbar.

Mit der Fokussierung auf den unterliegenden Orientierungsrahmen wird bei einer Dokumentarischen Interpretation ein Verstehen technisch-wissenschaftlicher Texte in einer anderen als der normalerweise in technischen und naturwissenschaftlichen Disziplinen gemeinten Weise angestrebt. Statt die im Text gegebenen Argumentationen nur nachzuvollziehen, um z. B. einen Beweis, die Funktionsweise eines technischen Geräts oder den Aufbau einer Programmiersprache zu erfassen, wird ein Verstehen im Sinne Webers angestrebt: aus dem Text werden die impliziten Wissensbestände und Voraussetzungen zumindest partiell erarbeitet, vor deren Hintergrund die expliziten Argumentationen Sinn ergeben. Um technischen oder wissenschaftlichen Fachtexten unterliegende Orientierungsmuster explizieren zu können, müssen diese Texte also über ihren fach- und wissenschaftsimmanenten Sinngehalt hinaus interpretiert werden; nur so können Teile des Orientierungsrahmens, die Verankerung des Textes in Selbstverständlichkeiten, Denkgewohnheiten und habitualisierten Handlungspraxen, sichtbar werden.

Ein zentrales Element der Dokumentarischen Interpretation ist die „komparative Analyse". Sie stellt eine Antwort auf das Problem der Standortgebundenheit der Interpreten dar, das sich – stellt man den Begriff einer absoluten, so gegebenen Wahrheit und damit zusammenhängend einer normativen Richtigkeit in Frage – in besonders prägnanter Weise stellt. Die Interpretation eines Textes erfolgt zunächst von dem eigenen Standort aus; vor dem Hintergrund des eigenen Vergleichshorizontes werden andere Handlungs- und Ausdrucksmöglichkeiten als in dem Text gewählt imaginiert (Bohnsack 1999):

> „Interpretation vollzieht sich grundlegend vor dem Vergleichshorizont einer Imagination hypothetischer alternativer Praktiken. Zu diesem Zweck ist die Interpretin genötigt, zunächst auf eigenes, intuitiv verfügbares handlungspraktisches Wissen zurückzugreifen. Hierin ist – wesentlich fundamentaler als im theoretischen Wissen – die Standort*ge-* und die *Seinsver*bundenheit der Interpretation verankert."

Um die Standortgebundenheit der Interpreten zu relativieren, werden im Rahmen der komparativen Analyse die untersuchten Texte mit anderen Texten kontrastiert: beispielsweise solchen aus dem gleichen Zeitraum mit gleichem Themenbereich. Auch möglich ist, Texte desselben Autors oder verschiedene Textstellen, in denen die gleiche oder eine ähnliche Thematik behandelt wird, einander gegenüberzustellen. Zu den imaginativ-hypothetischen Vergleichshorizonten der Interpreten treten so empirisch fundierte. So stellt sich, um ein einfaches Beispiel anzuführen, die Verwendung der Metapher „der Computer ist ein Nervensystem" in einem Text ganz anders dar, wenn ich diese nur auf der Basis meines eigenen Vergleichshorizontes einordne und vor dem Hintergrund der von mir gelesenen kritischen Literatur als unzureichend betrachte. Wenn ich hingegen andere Texte der gleichen Zeit befrage und feststelle, daß sie alle diese oder eine ähnliche Metaphorik verwenden, stellt sie sich als zeittypisches Phänomen dar, das seinen Ursprung in einer zu der Zeit weit verbreiteten Anschauung des Menschen hat.

Die Dokumentarische Interpretation eines Textes beginnt mit der „Formulierenden Interpretation". Diese hat zum Ziel, einen Überblick über den thematischen Verlauf des Gesamttextes zu erhalten. Bei diesem Schritt geht es darum, textimmanent zu verstehen, was ausgesagt wird. Dazu wird die thematische Feingliederung des Textes nachvollzogen und die jeweilige Aussage knapp und thesenartig zusammengefaßt. Da technische und wissenschaftliche Texte in der Regel recht klar strukturiert sind, die thematische Gliederung von daher relativ problemlos zu fassen ist, geht es nicht zuletzt darum, in diesem Schritt den wissenschaftsimmanenten Beitrag zu erfassen: z. B. den Beweis zu verstehen oder die Beschreibung des Aufbaus und der Funktionsweise eines technischen Geräts oder einer Programmiersprache nachzuvollziehen. Am Ende dieser Phase steht die Auswahl der im folgenden weitergehend zu interpretierenden Passagen. Auswahlkriterien sind dabei

- die thematische Relevanz einer Passage für die eigene Fragestellung: Sollen z. B. Nutzerbilder erarbeitet werden, sollten Textstellen, in denen es um mögliche Nutzer geht, weiterinterpretiert werden;
- die Vergleichbarkeit mit Textstellen aus anderen Papieren, die ebenfalls Teil der eigenen Untersuchung sind. Diese sind Voraussetzung für die oben beschriebene komparative Analyse;
- die besondere metaphorische Dichte: Werden in einer Textpassage besonders viele Metaphern (die zu der entsprechenden Zeit noch nicht üblicher Fachjargon sind) und bildliche Darstellungen verwandt, so verweist das darauf, daß hier neue Gedanken, die noch nicht in technisch-wissenschaftlicher Fachsprache ausgedrückt werden können, entwickelt werden;

- die stilistische Kompliziertheit: Unklare, verwickelte und schwer nachvollziehbare Beschreibungen und Argumentationen, die meist mit einem auffällig komplizierten Satzbau einhergehen, verweisen darauf, daß hier für den Autor schwierige und häufig neue Zusammenhänge herausgearbeitet werden. Im Gegensatz zu sehr klaren und stringenten Beschreibungen werden gerade in solchen Textstellen, über die man normalerweise gern hinwegliest, Brüche deutlich, die für ein Verständnis unterliegender Orientierungsmuster zentral sein können.

Den zweiten Schritt der Dokumentarischen Interpretation bildet die „Reflektierende Interpretation". Hierbei geht es darum, den Orientierungsrahmen zu rekonstruieren, innerhalb dessen das jeweilige Thema abgehandelt wird. Zentral ist in dieser Phase, zu unterscheiden zwischen einerseits Orientierungsschemata – expliziten Absichtserklärungen, Zielvorstellungen und Argumentationsketten der Autoren – und andererseits dem, was sich über die verwendete Begrifflichkeit und den Stil des Textes ausdrückt und auf den Orientierungsrahmen verweist. Die Absicht der Sprecher begegnet uns in den Texten lediglich dort,

> „wo die Erforschten selbst explizit über ihre Intention Auskunft geben, also auf der Ebene der Theorie *über* das eigene Handeln. Diese ist kategorial zu unterscheiden von jener Ebene der Handlungspraxis, der Realität des modus operandi, des Habitus, wie er dem Prozeß der Herstellung dieser Realität als implizites Wissen zugrundeliegt und sich an diesem Prozeß der Herstellung selbst dokumentiert." (Bohnsack 1999)

Im Rahmen der reflektierenden Interpretation geht es darum, implizite Wissensbestände herauszuarbeiten und ihren Bezug zu den explizit benannten Interpretationen und Argumentationen herzustellen. Die Ebene des impliziten Wissens und des Habitus gelangt dabei

> „primordial in metaphorischer und ‚atheoretischer' Weise [...] zum Ausdruck, nicht in begrifflich-theoretischer Explikation. Zum anderen (performatorisch) werden sie in der Handlungspraxis des Diskursprozesses entfaltet." (Bohnsack 1997, S. 200)

Um den Orientierungsrahmen zu rekonstruieren, ist also zum einen die Metaphorik des Textes zentral. Zum anderen muß die Handlungspraxis des Diskursprozesses betrachtet werden. Übertragen auf technisch-wissenschaftliche Texte bedeutet das die Frage danach, wie der Text geschrieben ist, und in welcher Weise das Thema abgehandelt wird, d. h. die Frage nach dem Stil des Textes. Im folgenden Abschnitt gehe ich deshalb darauf ein, wie der Stil und die Metaphorik eines wissenschaftlich-technischen Textes sowie deren Bedeutung herausgearbeitet werden können.

Die dokumentarische Interpretation schließt ab mit der Fall- oder auch Diskursbeschreibung. Diese soll auch für nicht mit dem interpretierten Text vertraute Personen lesbar und für eine eventuelle Veröffentlichung geeignet sein. Sie verdichtet die Ergebnisse und stellt die eigene Interpretation anhand charakteristischer Passagen des Textes dar, die beispielhaft für die in dem Text gewählte Sprachebene und Diskurs-

organisation sind. Anhand von Textsequenzen werden hier Komponenten des Orientierungsrahmens sowie unterliegende Orientierungsmuster entfaltet. Die in Kapitel 2 bis 4 dieser Arbeit enthaltenen Textbesprechungen stellen Fall- bzw. Diskursbeschreibungen dar, in denen zu Gunsten der besseren Lesbarkeit auf eine detaillierte Beschreibung der verwendeten interpretatorischen Mittel verzichtet wird.

Wichtige Elemente im diskursiven Herstellungsprozeß von Wirklichkeit und damit bei der Rekonstruktion des Orientierungsrahmens im Rahmen der Dokumentarischen Interpretation sind der Stil und die Metaphorik eines Textes, auf deren Untersuchung ich im folgenden eingehen werde.

Für den Bereich technisch-wissenschaftlicher Texte gibt es zur Bedeutung einzelner Stilelemente und Metaphernformen noch keine umfassenden, sondern nur partielle Untersuchungen. In „Moderne, Sprache, Mathematik" z. B. befragt Herbert Mehrtens die Art und Weise, wie mathematische Texte geschrieben sind. Als ein zentrales Stilmittel mathematischer Texte arbeitet er die demiurgische Attitüde des mathematischen Schöpfers heraus, in dessen „Sei M eine Menge“ sich seine Herrschaft über die Zeichen und auch das Gewaltsame der Mathematik dokumentiert (S. 459f.): „Die Sprache der Zeichen ist gewaltsam, weil die Zeichen gesetzt werden. Die mathematische Sprache hält sich an das, was unwiderruflich gesetzt werden kann. So kann sie Gesetze formulieren.“

In „Die Logik und das Schweigen“ untersucht Käthe Trettin – deren Arbeit hier als weiteres Beispiel für eine Analyse des Stils technisch-wissenschaftlicher bzw. formal gehaltener Texte dienen soll – die Evidenz, die logische Deduktionen und Formalismen in Texten produzieren, anhand des Aristoteles-Textes „Erste Analytik“ und Freges „Begriffsschrift“. Unterschlagen wird, so eines ihrer zentralen Ergebnisse, durch das Form-Faszinosum zum einen „die Zeitproblematik – und damit die universale und zugleich individuelle Endlichkeit des Lebens –“ sowie die Geschlechterdifferenz, die verschwiegen, verdrängt oder asymmetrisch bewertet wird (S. IX). Sowohl Mehrtens als auch Trettin schließen also von dem Stil des Textes auf den unterliegenden Orientierungsrahmen. Sie ermöglichen so, den expliziten Argumentationen der Texte unterliegende implizite Motivlagen zu thematisieren und damit die kulturelle Eingebundenheit und Gewordenheit der untersuchten Texte und Theorien neu zu reflektieren.

Um den Stil eines technischen oder wissenschaftlichen Textes herauszuarbeiten, kann zum einen die generelle Art der Argumentation und Beschreibung untersucht werden. Betrachtet werden sollte, wie begründet und wie detailliert dargestellt wird, wie Tatsachen als wahr oder gültig hergeleitet werden, und wie Evidenz hergestellt wird.

Dabei ist besonders die Rolle von Formeln, formalen Ableitungen, Tabellen, große Zahlen und graphische Abbildungen zu befragen.

Zum anderen ist hilfreich, den Bezugsrahmen des Textes – die Art und Weise, wie sich der Text in die Welt einordnet – zu analysieren. Dazu kann nach dem imaginierten Publikum und die diesem zuerkannten Kompetenzen, dem Bild der von der Theorie oder Technik betroffenen Gruppen (Nutzer oder Beforschte) sowie nach der Art der Darstellung der vorgestellten Theorie oder Technik gefragt werden:

- Wird die Technik oder Theorie als gemachtes, als geschichtliches Artefakt sichtbar?
- Stellt sie sich als vorexistent und nur Entdecktes dar?
- Wie eigenständig erscheint ein technisches Gerät?
- Wie werden Mensch und Maschine oder auch Theorie zueinander in Beziehung gesetzt?

Aufschlußreich ist auch die Selbstbefragung oder Selbstdarstellung der Autoren. Als wichtiges Stilmittel wissenschaftlich-technischer Texte fällt in diesem Zusammenhang auf, daß diese Texte bis heute von den Spuren ihrer Urheber im wesentlichen bereinigt sind – z. B. durch passivische Konstruktionen oder die Formulierung „man". Leider ist dies zumeist unabdingbare Voraussetzung, damit der Text überhaupt als ernstzunehmender wissenschaftlicher Beitrag in einer Community oder einem Denkkollektiv akzeptiert wird. Obwohl die Aspekthaftigkeit und Perspektivität von Wissenschaft und Technik theoretisch außer Frage steht, dokumentiert sich in dieser Bereinigung ein Habitus, den Blum in einem Artikel zu Fälschungen in der Wissenschaft in der ZEIT wie folgt beschreibt (Blum 1998, S. 36):

> „Trotz ihres subjektiven Verhaltens verstehen sich viele Wissenschaftler als Vertreter der absoluten Wahrheit, unabhängig von Zeit, Ort und Person. Aufgrund dieses Selbstverständnisses werden in den Publikationen alle Hinweise auf Emotionen wie Hoffnungen, Enttäuschungen und Entdeckerfreuden ausgelassen. Das literarische Rahmenwerk der wissenschaftlichen Publikation dient demnach der Mystifizierung eines durchaus menschlichen Unternehmens, und Mystifizierungen sind immer gefährlich."

Die Auslassungen in bezug auf den Autor und damit zusammenhängend auf die Konstruktions-, Entdeckungs- oder Erfindungsgeschichte bewirken in der Regel, daß die dargestellten Theorien oder auch Techniken den Charakter ewiger Gültigkeit erhalten, da ihre Geschichtlichkeit, Gewordenheit und Gebundenheit an ihre Entwickler in dem Text nicht sichtbar werden. Insofern dokumentieren gerade diese Auslassungen einen Denkstil, der die Kulturalität und Sozialität von Wissenschaft und Technik ausklammert.

Neben dem Stil stellt die Metaphorik eines Textes einen wichtigen Schlüssel zur Erarbeitung des unterliegenden Orientierungsrahmens dar. Donna Haraway macht die Wichtigkeit von Metaphern auch für technisch-wissenschaftliche Prozesse deutlich (Haraway 1981/82, S. 245):

„My purpose is to show how a natural object is constituted in scientific practice for us; i. e. made into a technical object with a knowable structure about which fruitful questions may be asked within the ordinary discipline of the appropriate natural scientific communities. This technical-natural object can only be constituted on the basis of material practice composed of the daily elements of scientific work: field and laboratory custom, machinery, social hierarchies and networks, funding possibilities, contests for priviledged model systems, metaphor, struggels over language."

Die Metaphorik eines Textes ist also ein wichtiges Element bei der Herstellung dessen, was überhaupt als technisches und wissenschaftlich verhandelbares Objekt in einer Community angesehen wird. Metaphorik hat in Technik und Wissenschaft einen brisanten politischen Charakter, denn sie strukturiert ein Wissensfeld und die Fragen, die hier gestellt und bearbeitet werden können, in subtiler Weise vor. Evelyn Fox Keller beurteilt Metaphern entsprechend nach ihrer Effektivität, d. h. nach der produktiven Wirkung, die sie in einem Wissensfeld entfalten (Keller 1995, S. XIff.):

„Needless to say, not all metaphors are equally useful or, for that matter, equally captivating. The effectiveness of a metaphor [...] depends on shared social conventions and perhaps especially, on the authority conventionally granted to those who use it. [...] Of course, the particular effectiveness of scientific metaphors depends not only on available social resources but also on the technical and natural resources that are available. Language does not simply construct reality."

Ob eine Metapher und das mit ihrer Verwendung grob vorgezeichnete Forschungsprogramm in einer Community angenommen wird oder nicht, hängt demnach von einer ganzen Reihe von Faktoren ab: von dem Stand der Forschung und der Technik, zu dem die Metapher passen muß und den sie verspricht, voranzubringen; von der Autorität derjenigen, die sie in den Diskurs einzubringen versuchen; von bereits akzeptierten disziplinären Konventionen, zu denen die Metapher nicht in Widerspruch geraten darf.

Auch im Bereich der Metaphernforschung in Wissenschaft und Technik ist noch viel Arbeit zu leisten (Keller 1995, S. XIV): „[...] it should be evident that a great deal more needs to be done to sort out the complex lines of influence and interactions of cultural norms, metaphor, and technical development." Ich gebe im folgenden einige Anregungen dazu, wie die Erfahrungshaltigkeit und Standortgebundenheit von Metaphern in Rahmen der Interpretation technischer oder wissenschaftlicher Texte erarbeitet, und somit auf den Orientierungsrahmen rückgeschlossen werden kann. Wichtige Voraussetzung ist dabei, die verschiedenen Wirk- und Funktionsweisen metaphorischer Prozesse benennen zu können. Ich werde deshalb einige auch für technische und wissenschaftliche Texte wichtige Metapherngruppen vorstellen. Dabei lehne ich mich wesentlich an Lakoff und Johnson (1980) an, die die Wirkungsweise von Metaphern allgemein untersucht haben.

Das Wort Metapher bezeichnet das Phänomen, ein Ding oder eine Gegebenheit mit Hilfe eines aus einem anderen Bereich stammenden Begriffs zu verstehen und zu erfahren. Dabei werden bestimmte Aspekte des metaphorisch bezeichneten Dings

(oder der Gegebenheit) hervorgehoben, andere hingegen der Wahrnehmung entzogen. Mit der Akzeptanz einer Metapher werden bestimmte Frage- und Vorstellungskomplexe in bezug auf das bezeichnete Ding selektiert (Lakoff, Johnson 1980, S. 13):

> „Thus, if ideas are objects, we can dress them up in fancy clothes, juggle them, line them up nice and neat, etc. So when we say that a concept is structured by a metaphor, we mean that it is partially structured and that it can be extended in some ways but not others."

Metaphern, die auf diese Weise wirken, d. h. die ein Konzept durch ein anderes strukturieren, nennen Lakoff und Johnson „strukturelle Metaphern". Für technische und wissenschaftliche Diskurse spielen solche Metaphern eine entscheidende Rolle, denn mit einer solchen Metapher werden bestimmte Sichtweisen auf den zu untersuchenden Gegenstandsbereich importiert. Die Erkenntnis, die in diesem Gebiet möglich ist, wird dadurch vorstrukturiert. Ein in der Informatik sehr bekanntes Beispiel ist, den Computer als intelligent zu bezeichnen – ein Prozeß, der die Ausbildung einer eigenen Forschungsdisziplin, der „Künstlichen Intelligenz", nach sich zog. Dieses Fachgebiet deckt die Fragen ab, die sich im Anschluß an diese Metapher eröffnen, z. B.: Wie schaffe ich es, daß der Computer eigenständig entscheidet, Sprache versteht, Expertenwissen besitzt usw. In den folgenden Kapiteln werde ich besonders zwei strukturelle Metaphern und ihre Wirkung auf den Bereich der Programmierung herausarbeiten: die des Computers als Nervensystem bzw. Gehirn und die, nach der das Programmieren in einer Sprache erfolgt.

Die in einer Gesellschaft oder in einer Gruppe verwendete Technik wird also auf der einen Seite mit Begriffen aus anderen Bereichen, und insbesondere dem der menschlichen Erfahrungen, Charakteristika und Aktivitäten beschrieben und begriffen (wie z. B. der Computer als Gehirn).[14] Umgekehrt prägt die so metaphorisch strukturierte Wahrnehmung dieser Dingtechnik die Erfahrungswelt und auch die Selbsterklärung der mit ihr umgehenden Personen. Unsere jeweiligen Erfahrungen mit physischen Objekten – und damit nicht zuletzt die in einer Gesellschaft, Kultur oder einem Kollektiv gebräuchliche Dingtechnik – bieten die Basis dafür, nicht-physikalische Erfahrungen wie Ereignisse, Aktivitäten, Gefühle oder Ideen zu begreifen und zu strukturieren. Lakoff und Johnson nennen Metaphern dieser Art „ontologische Metaphern". Solche Metaphern verweisen – und das ist im Kontext von technischer Wissenschaft besonders wichtig – auf spezielle Erfahrungen mit Dingen und besonders auch mit technischen Dingen, die durch metaphorische Prozesse in den Bereich menschlichen Denkens, Handelns und Fühlens übertragen werden. Ein in der theoretischen Informatik bekanntes Beispiel sind Automaten, also Maschinen, die z. B. für die Aktivität des Erkennens einer formalen Sprache stehen können. Diese Sicht auf den Spracherkennungsprozeß erlaubt, ihn wie eine Maschine in einzelne, klar

---

14 Lakoff und Johnson nennen aus dem Bereich menschlicher Charakteristika stammende Metaphern „personifizierende Metaphern".

abtrennbare Teile zu untergliedern, die in einer bestimmten zeitlichen Reihenfolge aktiviert werden. Auf die Automatenmetaphorik gehe ich in Kapitel 3 näher ein. Lakoff und Johnson selbst führen als Beispiel das nicht nur in der Informatik zentrale Bild „mind is a machine" an (Lakoff, Johnson 1980, S. 28):

> „The MACHINE metaphor gives us a conception of the mind as having an on-off state, a level of efficiency, a productive capacity, an internal mechanism, a source of energy, and an operating condition. The BRITTLE OBJECT metaphor [mind is a brittle object] is not as nearly as rich. It allows us to talk only about psychological strength."

Die Maschinenmetapher wirkt also dadurch inspirierend, daß sie erlaubt, das menschliche Denkvermögen bzw. den menschlichen Geist als operationalisierbaren Bereich zu fassen. Im Anschluß an diese Metapher ergeben sich eine ganze Reihe von Forschungsfragen und -strategien. Das macht sie in den Wissenschaften, die mit der Erforschung des menschlichen Denkens befaßt sind, zu einer äußerst produktiven Metapher.

Die Benennung einer Technik hat also weit größeren Einfluß, als nur das mit ihr zusammenhängende Forschungsprogramm vorzustrukturieren. Aus ihr können sich Selbst- und Menschenbilder sowie Erklärungen ableiten, die in ganz anderen gesellschaftlichen Bereichen angesiedelt sind. Foucault sprach deshalb auch von einer „Politik der Benennung", der im Angesicht des oben beschriebenen metaphorischen Zirkels kaum genug Bedeutung zugemessen werden kann. Die Metaphorik eines technischen oder wissenschaftlichen Textes bietet entsprechend ein reichhaltiges Feld, von dem aus Verbindungen zu anderen wissenschaftlichen Disziplinen, zu dem Text unterliegenden Weltsichten und Rationalitätsvorstellungen und zu Quellen, aus denen sich neue Ideen im Rahmen des Geneseprozesses einer Theorie oder Technik speisten, aufgezeigt werden können. Auch Erklärungen für finanzielle oder publizistische Unterstützung oder für unerwartet breite Popularisierung der wissenschaftlichen Inhalte können von hier aus gegeben werden.

Einerseits werden also technische Artefakte durch Begriffe aus der alltäglichen Erfahrungswelt, andererseits die alltägliche Erfahrungswelt durch technische Artefakte metaphorisiert. Bei der Interpretation eines technischen oder wissenschaftlichen Textes ist von daher zentral zu fragen, was wodurch metaphorisiert wird: der Mensch durch die Technik oder die Technik durch den Menschen. Besonders wenn dieses Verhältnis unklar ist – in einem Satz der Mensch durch Technik und wenig später die Technik durch den Menschen metaphorisiert wird –, wird die Begründungsbedürftigkeit einer Metaphorisierung in der Regel umgangen. Kritische Fragen, die mit der Verwendung der Metapher einhergehen könnten, fallen so unter den Tisch. Wie immer, wenn eine Metapher ohne Begründung eingeführt wird, dokumentiert sich eine spezifische Selektivität der Wahrnehmung, die auf den Orientierungsrahmen der Autors oder der Autorin schließen läßt.

In diesem Kapitel habe ich Ansätze aufgezeigt, die im Rahmen einer Dokumentarischen Interpretation technisch-wissenschaftlicher Texte dazu beitragen, die Sinngefüge, in die Technikerinnen und Wissenschaftler ihre Handlungen und Produkte einbetten, sichtbar zu machen. Eine solche Interpretation dient dazu, implizites Wissen – von Fleck in dem Begriff des Denkstils, von Foucault als positives Unbewußtes gefaßt – und seine handlungsleitende Kraft für eine Theorie- oder Technikentwicklung herauszuarbeiten. Es werden gewissermaßen Motivlagen und Faszinosa benannt, die für die Beforschten so selbstverständlich sind, daß sie nicht bemerkt werden (Bohnsack 1999):

> „Auf der Grundlage des impliziten Wissens der Beforschten selbst, d. h. auf der Grundlage ihrer Beschreibungen, Erzählungen und Diskurse, werden jene verallgemeinerbaren Regeln bzw. Orientierungsmuster rekonstruiert, die den Erforschten zwar wissensmäßig verfügbar sind, die sie aber selbst – je tiefer diese in ihrer habitualisierten, routinemäßigen Handlungspraxis verankert sind – um so weniger zu explizieren vermögen."

Obwohl es dabei nicht um die moralische Bewertung eines anderen Standortes geht, wirkt das Benennen von Selbstverständlichkeiten im Rahmen einer Wissenschafts- und Technikentwicklung häufig befremdlich und unangemessen. Das verweist darauf, inwiefern das Explizieren von Orientierungsmustern in der Technik- und Wissenschaftsentwicklung als Kritik aufgefaßt werden kann: Was durch ein solches Herangehen an Technik und Wissenschaft in Frage gestellt wird, ist der Glaube an eine objektive Rationalität oder an eine in dieser Hinsicht privilegierte Stellung von technisch-wissenschaftlichen Entscheidungen und technisch-wissenschaftlichem Handeln. Indem Orientierungsmuster, die Ausdruck der Sozialisationsgeschichte und kultureller, habitualisierter Handlungspraxis sind, als Faktoren der Wissenschafts- und Technikentwicklung sichtbar gemacht werden, können Technik und Wissenschaft nicht länger als von der Kultur und Gesellschaft prinzipiell unabhängige, objektiv rationale Entwicklungen begriffen werden. Technik und Wissenschaft erscheinen selbst als durch und durch kulturell und sozial.

Kapitel 2: Die Entstehung der Speicherprogrammierung

Die Entstehung der Programmierung und der Programmiersprachen läßt sich in verschiedenster Weise beschreiben – so z. B. als eine Geschichte der Maschinen, ohne die die moderne Programmierung nicht denkbar wäre,[15] der mathematischen Kalküle[16] oder der gesellschaftlichen Formalisierung[17]; und jede dieser Geschichten gewinnt ihre Berechtigung und ihren Erkenntniswert im Kontext eines bestimmten Anliegens, einer Frage, die beantwortet werden soll. Welches also ist das Anliegen dieser Arbeit, und welche Perspektive ergibt sich daraus? Wie im letzten Kapitel dargestellt, spielen in der wissenschaftlich-technischen Arbeit und Entwicklung Sinngeflechte eine zentrale Rolle, in die neue noch in der Formulierung begriffene Ideen und im Zusammenhang damit entwickelte Techniken eingebettet werden. Ich frage in dieser Arbeit deshalb nach den Orientierungsmustern, vor deren Hintergrund sich die Programmierung herausbildete und weiterentwickelte, und danach, wie diese Orientierungsmuster neue Ideen formierten und strukturierten. Dabei wird sich zeigen, daß diese in wesentlichen Teilen bis heute tradiert wurden und auch der Gestaltung neuerer Theorien und Techniken im Bereich der Softwareproduktion unterliegen. Obwohl der Bereich der Computertechnologie sich mit einer beachtlichen Dynamik entwickelt, bilden unterliegende Orientierungsmuster – und insbesondere die Hybridobjekt-Sichtweise – eine relativ konstante Struktur, die immer wieder reproduziert wird.

Die Hybridobjekt-Sichtweise war schon für den den „First Draft of a Report on the EDVAC" von 1945, die „Preliminary Discussion of the Logical Design of an Electronic Computing Instrument" von 1946 sowie die „Planning and Coding Problems for an Electronic Computing Instrument" von 1947[18] – die drei Texte, die Hauptgegenstand dieses Kapitels sind, – und damit am Beginn der Speicherprogrammierung zentral. Sie dokumentiert sich in der Art und Weise, wie der Autor, John von Neumann, die Rechenmaschine, den Code und das Kodieren hier darstellt. Er behandelt die Rechenmaschine nicht mehr nur als elektronische Maschine, deren Bau handwerklich-technisches Geschick, pragmatische Entscheidungen und den gewitzten Umgang mit elektronischen Bauteilen erfordert – eine Sicht und Kultur, die noch beim Bau des ENIAC, dem Vorläufer des EDVAC, bestimmend war. Die Rechenmaschine erscheint in dem „First Draft" und abgeschwächt in der „Preliminary Discussion" auch als eine Art Organismus, der dem menschlichen Nervensystem ähnelt. Das und eine Argumentationsweise, nach der sich der generelle Aufbau des Geräts aus seiner

---

15 Vgl. z. B. (Williams 1985), (Randell 1982).

16 Vgl. z. B. (Krämer 1988), (Mehrtens 1990).

17 Vgl. z. B. (Heintz 1993).

18 Im folgenden werde ich über diese drei Texten der Einfachheit halber als „First Draft", „Preliminary Discussion" und „Planning and Coding Problems" sprechen.

Funktion – Instruktionen zum Durchführen komplexer Berechnungen auszuführen – ergibt, erlaubt, von den materiellen Gegebenheiten zu abstrahieren; Rechnerbau erscheint damit nicht zuletzt als ein mathematisch-logisches Problem. Konkrete Rechner können als Spezialfälle gesehen werden, die eine übergeordnete Gesetzlichkeit materiell füllen.

Die drei Sichten auf das Gerät des "First Draft" – als Maschine, Organismus und gesetzliche Struktur – spiegeln sich in der Darstellung des Codes sowie des Kodierens wider: Auch der Code erscheint in den untersuchten Texten zunächst als technisches Artefakt und scheint die Maschine zu steuern. Wie die Rechenmaschine auch als eine Art elektronisches Nervensystem oder Gehirn gesehen wird, wird der Code jedoch auch als eine Liste von Befehlen interpretiert, die die Maschine versteht und ausführt. Da die Maschine im wesentlichen zu rechnen scheint wie ein menschlicher Rechner, scheint der Code dabei Ähnlichkeit mit der mathematischen Sprache zu besitzen. Die Sicht auf die Rechenmaschine als gesetzliche Struktur schließlich findet ihren Ausdruck in der Sicht auf das Kodieren, die Arbeit mit dem Code. Das Kodieren wird in den „Planning and Coding Problems“ nicht als technische, eng an die Maschine gebundene Tätigkeit konzipiert, obwohl die enge Abhängigkeit zwischen Code und Aufbau der Maschine an vielen Stellen deutlich wird. Statt dessen kann das Kodieren – da auch als Manipulation einer gesetzlichen Struktur denkbar – als mathematische Tätigkeit und neuartiges Aufgabengebiet der formalen Logik beschrieben werden.

Der „First Draft“, die „Preliminary Discussion“ und die „Planning and Coding Problems“ schufen eine Grundlage dafür, die Rechenmaschine und ihre Programmierung zu mathematisieren und zu verwissenschaftlichen – und nicht zuletzt in diesem Sinne steht von Neumann mit diesen Texten an den Anfängen einer neuen Wissenschaft, der Informatik. Seinen Texten unterliegt ein Orientierungsrahmen, der erlaubt, in der Rechenmaschine nicht nur eine spezielle Maschine und in der Kodierung nicht nur die Bedienung einer solchen Maschine zu sehen. In der Art und Weise, wie die Rechenmaschine und der Code hier konzipiert sind, dokumentiert sich, daß die Grenzen zwischen technisch und biologisch und in einem zweiten Schritt – entsprechend dieser Irrelevanz der materiellen Umsetzung – zwischen dem konkreten Ding und einem Symbolsystem im Diskurs über Rechenmaschinen und ihre Programmierung an Wichtigkeit verlieren. Die in den von Neumannschen Arbeiten deutlich werdende Sichtweise erlaubt, die Informatik als eine Wissenschaft zu konzipieren, in der es nicht nur um eine Maschine, sondern um Informationsverarbeitung allgemein geht.

2.1 Vorgeschichte: Von Neumann, das ENIAC- und das EDVAC-Projekt

Wenn in diesem und in dem folgenden Kapitel Texte, die John von Neumann allein oder in Zusammenarbeit mit anderen verfaßte, im Zentrum stehen, dann vor allem deshalb, weil sie für die Art und Weise, wie die Rechenmaschine und ihre Programmierung wahrgenommen wurde und bis heute wird, eine zentrale Rolle spielen. Sie waren zumindest im englischen Sprachraum die meistgelesenen Texe zum Thema Speicherprogrammierung und wurden auch im deutschen Sprachraum breit rezipiert. Entsprechend wird von Neumann heute eine, wenn nicht die zentrale Rolle bei der Beantwortung der Frage nach den Anfängen der Informatik und Programmierung zugewiesen. Darauf verweist schon, daß bis heute der Aufbau der meisten modernen Rechner als „Von-Neumann-Architektur" bezeichnet wird, so daß jede und jeder, falls näher mit Computern befaßt, zumindest mit seinem Namen bekannt ist. Nicht selten wird die Erfindung des modernen Rechners denn auch auf von Neumann zurückgeführt – eine Darstellung, die sich zwar bei näherem Hinsehen als stark verkürzt erweist, aber trotzdem weit verbreitet ist. Schon 1956 hieß es in dem Spiegelartikel „Elektronengehirne. Die Magie der Roboter" unter der Zwischenüberschrift „Photozellen statt Augen" (Spiegel, 3. Oktober 1956, S. 50):

> „John von Neumann, einer der brillantesten Mathematiker der Welt, Professor in Princeton und Mitglied der amerikanischen Atomenergiekomission, half dem Denken der immer größeren, immer schnelleren Elektronen-Giganten, die nach der Erstgeburt ‚Eniac' entstanden, buchstäblich auf die Sprünge."

28 Jahre später wird im Duden der Informatik unter dem Eintrag „Von-Neumann-Rechner" von Neumann als der alleinige Erfinder dieses Konzepts angeführt:

> „Im Jahre 1946 von John von Neumann vorgeschlagenes Konzept zur Gestaltung eines universellen Rechners [...]. Von einigen wenigen Ausnahmen abgesehen [...], orientieren sich die heutigen Rechenanlagen an der Struktur dieses klassischen Universalrechners."

Auch wenn es darum geht, die Anfänge der Wissenschaft Informatik als Ganzes zu ergründen, wird nicht selten von Neumann ein Ehrenplatz zugewiesen. Friedrich L. Bauer z. B., einer der wichtigsten Protagonisten der Etablierung der Informatik an bundesdeutschen Hochschulen, setzt sich in seinem Text „Was heißt und was ist Informatik?" 1974 mit der Frage auseinander, wer als Urheber der Informatik gelten dürfe. Den Ingenieur Konrad Zuse hält er dabei zwar für herausragend; da dessen Pläne zur Programmierung aber zu weitgehend und daher noch nicht realisierbar gewesen seien, komme er für diesen Ehrentitel nicht in Frage. Bauer schreibt (Bauer 1974, S. 336):

> „Somit hätte auch ein Ingenieur der Begründer der Informatik sein können. Die überragende mathematische Begabung, das Gespür für Wesentliches wie auch die beherrschende Stellung im Wissenschaftsbetrieb ließen aber J. von Neumann zu dieser Rolle kommen [...]".

1984 stellt ähnlich Bernd Mahr, Informatikprofessor an der Technischen Universität Berlin, fest (Mahr 1984, S. 100 f.):

„Schließlich sieht er [von Neumann] den Bau komplexerer Automaten nur dann als möglich an, wenn wir eine sehr fortgeschrittene und subtile Theorie der Automaten und der Information besitzen. [...] Da hat also vor 36 Jahren einer ausgesprochen, was sich uns heute noch, und durch die Entwicklung hochintegrierter Schaltkreise wieder, in besonderem Maße als Problem stellt. [...] Er gilt als der Vater der modernen Computer, deren Bauprinzipien sich auf seine Veröffentlichungen gründen."

Auch über den deutschen Sprachraum hinaus gilt von Neumann als einer der Begründer der Computerwissenschaften, dessen Ideen bis heute Einfluß besitzen. Er gilt als überragendes mathematisches Genie und wird z. B. in einem bekannten US-amerikanischen Buch zur Geschichte der Künstlichen Intelligenz als „intellectual giant" (McCorduck 1979, S. 62) oder von dem Ungar Ádám als „eines der seltenen Genies, die ihre umfassenden Interessen, ihre Phantasie, ihre ungeheure Arbeitskraft zu den universellen, führenden Persönlichkeiten dieser Wissenschaft [Mathematik] erheben" (Ádám 1979, S. 11) gefeiert. Ebenfalls in Ungarn, dem Geburtsland von Neumanns, schreibt Szentiványi (Legendi, Szentiványi 1983, S. 5):

„Vor kurzem wäre John von Neumann, einer der vielseitigsten Informatiker unserer Zeit, 75 Jahre alt geworden. [...] Seine Lösungen und Ergebnisse sind von grundlegender Bedeutung, seine Gedanken wirkten und wirken seitdem befruchtend."

In dem selben Band stellt der US-Amerikaner Goldstine, ein ehemaliger Kollege und Bewunderer von Neumanns, der nach dessen Tod eine ganze Reihe vielgelesener Texte zur Rolle von Neumanns in der Rechnerentwicklung verfaßte, fest (Goldstine 1979, S. 45):

„Obwohl mehrere, vielleicht ein Dutzend Menschen eine wesentliche Rolle bei der Entwicklung der Rechner spielten, müssen wir doch von Neumann die wichtigste zugestehen. [...] Hier ist nur davon die Rede, daß jene Arbeit, die von Neumann in der Rechnerentwicklung leistete, meiner festen Überzeugung nach qualitativ besser, tiefergehender und von größerer Wirkung war als die Leistungen eines jeden anderen."

Ähnlich schreibt Burks, ebenfalls ein ehemaliger US-amerikanischer Computer-Pionier, der sich später in der Geschichtsschreibung der Computerentwicklung engagierte (Burks 1987c, S. XI):

„Von Neumann's writings are fundamental to the historian or computer professional who wants to gain a historical perspective on computer science. [...] These writings also have value to contemporary computer research."

Von Neumann, seine Arbeit und seine Veröffentlichungen nehmen also in der Geschichte der Informatik und Rechenanlagen, die ja wesentlich von Computerwissenschaftlern selbst geschrieben wird, und damit auch im modernen Selbstverständnis der Computerwissenschaften einen zentralen Platz ein. Darüber hinaus wird die Wirkung, der Einfluß und auch die Aktualität seiner Ideen bis heute immer wieder betont. Als Teil dessen, was heute zumeist von Informatikerinnen und Informatikern selbst als Tradition der Informatik und der Programmierung konstruiert wird, sagen seine Texte vielleicht sogar mehr über das heutige Selbstverständnis der Informatik und Computerwissenschaft als über die Vergangenheit. Es liegt nahe, daß die Orientierungsmuster, die ihnen unterliegen, auch für den heutigen Blick auf die Rechen-

maschine und die Programmierung sowie das Sinngeflecht, in das diese eingebettet werden, zentral sind. Von Neumanns Ideen und Papiere – so meine noch auszuführende These – waren entsprechend der überragenden Bedeutung, die ihnen bis heute zugeschrieben wird, nicht nur in einem rein technischen Sinne richtungsweisend für die Entwicklung von Rechenmaschinen und ihrer Programmierung. Wichtig waren sie auch dafür, wie diese Maschinen und ihre Programmierung in der Folgezeit und bis heute wahrgenommen und in Sinnzusammenhänge eingebettet wurden, wie über sie geredet und nachgedacht wurde. Da von Neumann Zusammenhänge zwischen verschiedenen wissenschaftlichen Bereichen wie Mathematik, Ingenieurwissenschaften, Neurophysiologie und Genetik formulierte, wird in seinen Texten die Verortung der neuen Rechnertechnologien im Wissen der Zeit besonders deutlich. Von Neumanns Texten zu Rechenanlagen und ihrer Programmierung unterliegen Orientierungsmuster, die die Visionen und schließlich die Gestalt nachfolgender Theorien und Techniken bis heute prägen.

Als von Neumann im Laufe des zweiten Weltkrieges begann, sich für Rechenmaschinen zu interessieren, war er bereits ein international bekannter und im amerikanischen Wissenschaftsbetrieb außerordentlich angesehener und einflußreicher Mathematiker. Er hatte zahlreiche Kontakte zu Wissenschaftlern verschiedenster Disziplinen. Sein Wort hatte Einfluß, und er hatte, wie wohl nur wenige andere, einen guten Überblick über wissenschaftliche Entwicklungen auch in anderen Disziplinen als der Mathematik:

Als Kind wohlhabender Eltern am 28. 12. 1903 in Budapest geboren, war von Neumann besonders in Mathematik seit der frühen Kindheit stark gefördert worden.[19] 1925 erwarb er nach Studien in Berlin, Budapest und Zürich den Abschluß des Chemieingenieurs in Zürich, 1926 legte er die mathematische Doktorprüfung in Budapest ab. Von 1926 bis 1929 war er als Privatdozent an der Berliner und von 1929 bis 1930 an der Hamburger Universität tätig. Zudem besuchte er Seminare an der Universität Göttingen, dem damaligen Zentrum der modernen Mathematik. 1930 ging er zunächst als Gastdozent nach Princeton und arbeitete bis 1933 zur Hälfte in Berlin und zur Hälfte in Princeton. Als ihm 1933 vom „Institute of Advanced Studies“ in Princeton eine Professur angeboten wurde – die Stelle, die er bis zu seinem Tod am 8. 2. 1957 behalten sollte[20] –, siedelte er auf Grund der sich für Juden in Deutschland und Europa immer weiter verschlechternden Lage in die USA über. Während und auch

---

19 Die folgende kurze Darstellung von von Neumanns Biographie stützt sich auf (Ádám 1979), (Burks 1987b), (von Neumann, Klara 1958). Eine sehr ausführliche und kritische Darstellung seines Lebensweges gibt (Heims 1987).

20 Von Neumann verstarb an Knochenkrebs, was Anlaß zu der Spekulation gab, daß er im Rahmen seiner Arbeit an der Atom- und später der Wasserstoffbombe zu sehr radioaktiver Strahlung ausgesetzt war.

nach dem Zweiten Weltkrieges war er Berater in einer ganzen Reihe militärischer Forschungsprojekte (Burks 1987a, S. 3):

> „Altogether, von Neumann was involved in such diverse fields as ordnance, submarine warfare, bombing objectives, nuclear weapons (including the hydrogen bomb), military strategie, weather prediction, and intercontinental ballistic missiles."[21]

Insbesondere war er mathematischer Berater in dem Atombombenprojekt in Los Alamos. Nachdem er sich bereits vor dem Krieg mit der Theorie der Schock- und Detonationswellen auseinandergesetzt hatte, erhielt er diesen Posten auf Wunsch von J. Robert Oppenheimer, den er aus seiner Göttinger Zeit persönlich kannte. Seine Hauptaufgabe bestand hier darin, zusammen mit Edward Teller und anderen eine Zündungstechnik für die Bombe zu entwickeln, die sog. „implosion method", die später auch für die Zündung der Nagasaki-Bombe eingesetzt wurde. Bei dieser wurde eine Kugel Plutonium- oder Uran-Isotope so kritisch zusammengepreßt, daß die atomare Kettenreaktion in Gang gesetzt wurde (vgl. Slater 1987, S. 26f.). Hagen beschreibt die mathematische Problematik dieser Zündungstechnik wie folgt (Hagen 1989, S. 215):

> „Das mathematische Problem bestand in der Gewinnung einer Formel für eine Druckwelle, die in exakter Simultaneität alle Punkte der nuklearen Masse gleichermaßen erreicht; – und es ging darum, für diese Formel in Zahlen ausdrückbare Lösungen zu finden."

Nötig war damit – wozu von Neumann entscheidend beitrug – die Erarbeitung geeigneter numerischer Verfahren. Der anfallende numerische Rechenaufwand verstärkte den Bedarf an Rechenleistung enorm und war ein, wenn nicht der Anlaß für von Neumanns Interesse an Hochgeschwindigkeitsrechenanlagen und insbesondere dem ENIAC- bzw. EDVAC-Projekt.

Das Interesse an Rechenmaschinen teilte von Neumann mit dem ebenfalls sehr bekannten Mathematiker Norbert Wiener, mit dem ihn bereits vor dem Krieg eine fachliche und private Freundschaft verband. Wiener arbeitete in dem Zweiten Weltkrieg an einer speziellen Rechenmaschine, dem „antiaircraft (AA) predictor". Dieser sollte „den Zickzackflug eines feindlichen Flugzeuges bestimmen, seine künftige Position im voraus berechnen und eine Flugabwehrrakete starten, um es abzuschießen." (Galison 1997, S. 282) Für Wiener stellten sich in Zusammenhang mit dem Prädiktor nicht nur mathematische und technische Probleme; zusammen mit dem Ingenieur Julian Bigelow und dem Kardiologen Arturo Rosenbluth diskutierte er in

---

21 Auch nach dem Krieg blieb von Neumann gefragter Berater der Regierung und der Privatindustrie. Einen Überblick über seine zahlreichen Tätigkeiten nach dem Krieg gibt (Burks 1987b, S. xviii): „He worked closely with Ballistic Research Laboratory, Los Alamos Scientific Laboratory, Naval Ordnance Laboratory, the Research and Development Board, Oak Ridge National Laboratory, the Armed Forces Special Weapons Project, the Weapons System Evaluation Group, the US Air Force Scientific Advisory Board, the US Atomic Energy Comission, the Technical Advisory Panel on Atomic Energy, the Rand Corporation, International Business Machines, and Standard Oil Development Company."

dem Artikel „Behavior, Purpose and Teleology“[22] psychologische und philosophische Auswirkungen des Prädiktors.

Nachdem dieses Papier veröffentlicht war, sprach Wiener mit von Neumann über seine Ideen zu Analogien zwischen elektronischen Geräten und lebenden Organismen. Zusammen mit Howard Aiken organisierten sie im Anschluß ein Treffen einer kleinen Gruppe von Wissenschaftlern, „die an Kommunikationstechnik, an der Technik der Rechenmaschinen. an der Technik der Kontrollvorrichtungen, an der Mathematik der Zeitreihen in der Statistik sowie an Kommunikations- und Kontrollaspekten des Nervensystems interessiert sind.“ (Aus dem Einladungsbrief von Wiener, Aiken und von Neumann an S. S. Wilks, Pitts, E. H. Vestine, W. E. Deming, McCulloch, R, Lorente de Nó, Leland E, Cunningham, 4. 12. 44. Nach Galison 1997, S. 301, vgl. auch Heims 1987, S. 185). Nach der Zusammenkunft, die am 6. und 7. 1. 45 am „Princeton Institute“ stattfand, war die Gruppe entschlossen, nach dem Krieg eine neue Forschungsrichtung – die Wiener 1947 „Kybernetik“[23] nannte – zu begründen, was Wiener in einem anschließenden Brief Rosenbluth, der nicht teilnehmen konnte, wie folgt mitteilte: Das Treffen war, so Wiener hier

> „a great success. I believe you have already got von Neumanns report. [...] The first day von Neumann spoke on computing machines and I spoke on communication engineering. The second day Lorente de Nó and McCulloch joined forces for a very convincing presentation of the present status of the problem of the organization of the brain. In the end we were all covinced that the subject embracing both the engineering and neurology aspects is essentially one and we should go ahead with plans to embody these ideas in a permanent program of research [...]“ (nach Heims 1987, S. 185f.).

Von Neumann war also, bevor er den „First Draft“ niederschrieb, in lebhafte, interdisziplinäre Diskussionen zwischen Mathematikern, Technikern und Neurophysiologen um die Organisation des Gehirns und den Zusammenhang zwischen Ingenieurwissenschaften und Neurophysiologie involviert. Eine gemeinsame Überzeugung dieser Gruppe bestand darin, daß neurologische und ingenieurwissenschaftliche Fragen im wesentlichen ein Themengebiet sind – eine Auffassung, die historisch verstanden werden muß und sich, wie Galison (1997) am Beispiel Wieners zeigt, in dieser Form im Rahmen der kriegsbedingten Forschung herausbildete (vgl. Abschnitt 3.1.1. Exkurs). Diese Auffassung fand auch – wie in den folgenden Abschnitten deutlich werden wird – in von Neumanns Papieren ihren Ausdruck und spielt bis heute in der Wahrnehmung von Rechenmaschinen eine Rolle.

Von Neumann schrieb den „First Draft of a Report on the EDVAC“ in seiner Eigenschaft als Berater des EDVAC-Projektes, das ein Folgeprojekt des Baus des ENIAC

---

22 Auf diesen Artikel komme ich in Abschnitt 3.1.1 noch genauer zu sprechen.

23 Den Namen Kybernetik leitete Wiener 1947 von dem griechischen „kybernetes“, zu deutsch „Steuermann“, ab.

war. Um auch diesen Teil des Enstehungskontextes des „First Draft“ zu beleuchten, werde ich nun auf diese beiden Projekte genauer eingehen.

Der ENIAC (Electronic Numerical Intergrator and Automatic Calculator)war der direkte Vorläufer der EDVAC (Electronic Discrete Variable Automatic Computer)[24] . Mit ihm wurde die technische Grundlage dafür geschaffen, daß ein Projekt wie der EDVAC in dieser Form denkbar wurde. Außerdem entstand im Rahmen der ENIAC-Gruppe die Idee der Speicherprogrammierung, wie sie in wesentlichen Grundzügen bis heute aktuell ist. Zwar gibt es Spekulationen darüber, daß bereits Babbage zu Anfang des 19. Jahrhunderts Ideen zur Speicherprogrammierung entwickelt habe – sicher ist jedoch, daß diese auf die moderne Entwicklung keinen Einfluß hatten. Auch Zuse erwähnte 1936 die Idee der Speicherprogrammierung in einem Papier, verfolgte diese Idee jedoch zunächst nicht weiter. Denn (Randell 1982, S. 376): „In fact the idea of a stored program has little attraction when a machine has only a small mechanical internal memory.“ Erst wenn eine genügend effiziente Speichertechnologie zur Verfügung steht, ist also die Idee der Speicherprogrammierung so technisch realisierbar, daß sie einen praktischen Nutzen hat. Die Technologie aber, die benötigt wurde, um einen solchen großen und schnellen Speicher bauen zu können, wurde erst auf Grundlage der Erfahrungen des ENIAC-Projektes ausgearbeitet und entwickelt.

Ein Papier, das häufig mit der Enstehung des Konzepts der Speicherprogrammierung in Zusammenhang gebracht wird, ist Turings „On Computable Numbers with an Application to the Entscheidungsproblem“ von 1936.[25] Turings Beitrag gehörte jedoch in ein Teilgebiet der Mathematik, das erst später so eng mit dem tatsächlichen Bau von Rechenmaschinen in Verbindung gebracht wurde, wie das heute der Fall ist.[26] Turings Maschinen waren ein Vehikel, den Begriff der Berechenbarkeit formal zu fassen; ihr technischer Aufbau sowie die tatsächliche Ausführung maschineller Berechnungen spielten dabei keine Rolle. Daß Turing bei dem Versuch der Formalisierung des Berechenbarkeitsbegriffs auf die Beschreibung durch einen Mechanismus, eine Maschine, verfiel, verdankte sich nicht dem Wunsch, eine Rechenmaschine zu bauen, sondern läßt sich, wie Mehrtens herausarbeitet, zurückführen auf innermathe-

---

24 In der Literatur wird im Deutschen „the ENIAC“ bzw. „the EDVAC“ nicht einheitlich übersetzt: sowohl der männliche als auch der weibliche Artikel kommen vor. Ich werde im folgenden den männlichen Artikel wählen.

25 So ist Turings Prinzip der universellen Maschine häufig als Vorwegnahme des speicherprogrammierbaren Computers interpretiert worden. Davis z. B. schreibt (1995, S. 144): „The universal machine also provided a model of a ‚stored program‘ computer in which the coded quintuples on the tape play the role of a stored program, and in which the machine makes no fundamental distinction between ‚program‘ and ‚data‘.“

26 Wie Eulenhöfer (1998b, S. 83ff.) zeigt, spielte Turing im Diskurs zu Rechenmaschinen in Deutschland erst sehr spät eine Rolle. Er war damit nicht für die Rechenmaschinenentwicklung selbst, sondern für die später erfolgende mathematische Interpretation und Aneignung von Rechenmaschinen und ihrer Programmierung von Bedeutung.

matische Diskussionen um das Hilbertsche beweistheoretische Programm.[27] Mathematik und mathematisches Denken wurden mit Hilfe einer Metaphorik aus dem Bereich der Mechanik beschrieben (Mehrtens 1990, S. 302):

> „Hilberts Versuch einer Verbindung idealer und realer Wahrheit bedient sich ständig mechanischer Analogien. Es geht um den Bau des ‚Apparats' der Mathematik und die ‚Mechanik' des Denkens als einer Entscheidungsmaschine."

Turings Vorgehen, Rechnen als einen maschinellen Vorgang zu beschreiben, war also zunächst eine innermathematische Angelegenheit ohne Bezug zum Bau realer Rechenmaschinen. Trotzdem war Turings Papier für von Neumanns Arbeiten zumindest auf Umwegen wichtig.[28] Wie ich in Abschnitt 2.2.2 herausarbeiten werde, spielte er für die Art und Weise, wie von Neumann in dem „First Draft" die Rechenmaschine darstellte – als eine Art Organismus – auf dem Umweg über eine andere Arbeit (McCulloch, Pitts 1943) eine Rolle.

Der ENIAC wurde gegen Ende des Zweiten Weltkriegs an der „Moore-School of Electrical Engineering" der Universität von Pennsylvania in Philadelphia gebaut.[29] Seit den frühen 30er Jahren arbeitete die Moore-School mit dem „U.S. Army Ordnance Department Ballistic Research Laboratory" am Aberdeen Proving Ground in Maryland zusammen. Aufgabe des „Ballistic Research Laboratory" war es, Zieltabellen für die Artillerie zu erstellen. Die dazu nötigen Berechnungen wurden, wie das in dieser Zeit üblich war, von einer eigenen Rechenabteilung durchgeführt. Diese umfaßte 200 speziell ausgebildete Mathematikerinnen, die damals „computer" genannt wurden.[30] Diese Abteilung wurde von Leutnant Herman H. Goldstine, vor dem Krieg „Assistent Professor of Mathematics" an der Universität von Michigan und später Mitarbeiter von Neumanns, geleitet. Die Berechnung einer Flugbahn, durchgeführt von einer ausgebildeten Person mit einem elektrisch betriebenen Tischrechner, dauerte 20 Stunden – die Berechnung einer einzigen Tabelle ca. 30 Tage. Anfang 1943 hinkte die Erstellung ballistischer Tabellen deshalb so weit hinter den im zweiten Weltkrieg enorm gewachsenen militärischen Anforderungen hinterher, daß Goldstine beauftragt

---

27 Bis heute haben Turingmaschinen ihre Bedeutung im Rahmen der mathematischen Disziplin: Mitte der sechziger Jahre wurden Turingmaschinen zum allgemein akzeptierten Modell, um die Komplexität von Berechnungen zu messen (vgl. Mahoney 1992, S. 351).

28 Den verschiedenen Verbindungen (Treffen etc.), die zwischen Turing und von Neumann bestanden bzw. dem, was heute noch davon rekonstruierbar ist, spürt Davis (1995) eingehender nach. Ich werde mich darauf beschränken, die Texte von Neumanns und ihren Bezug zu (Turing 1936) zu betrachten.

29 Die folgende Darstellung des ENIAC-Projekts stützt sich wesentlich auf (Williams 1985, S. 271ff.). Williams geht neben dem technischen Aufbau auch auf den finanziellen Hintergrund, die Arbeitsatmosphäre und die beteiligten Personen ein und gibt eine gut recherchierte Darstellung sowohl der ENIAC- als auch der EDVAC-Entwicklung.

30 Vgl. dazu (Hoffmann 1987, S. 76ff.). Hoffmann untersucht in ihrer Arbeit die Rolle von Frauen bei der Programmierung des ENIAC und in der Computerarbeit allgemein.

wurde, nichts unversucht zu lassen, um diese Situation zu verbessern. Die Rechenbüros und die dort arbeitenden „computer“ sollten, so macht die weitere Geschichte deutlich, durch elektronische Rechenmaschinen ersetzt werden – ein in bezug auf die Frage der Sinnzusammenhänge, in die diese Maschine eingebettet wurde, wichtiger Faktor, auf den ich noch zu sprechen kommen werde.

Am 9. 4. 1943 fand ein Treffen zwischen der Moore School, die bereits zwei „differential analyser“[31] im Auftrag des Ballistic Research Laborartory gebaut hatte, und dem Ballistic Research Laboratory statt. J. Presper Eckert, ein Student der Elektrotechnik, und John Mauchly, bis zum Krieg Professor für Physik am Ursinus College bei Philadelphia, stellten hier die Idee zur Konstruktion eines „Electronic Numerical Integrator“ vor. Ihre Vorstellung beruhte auf einem kurzen, drei bis vier Seiten langen Papier – „The Use of High Speed Vacuum Tube Devices for Calculating“ – das Mauchly nach entsprechenden Diskussionen mit Eckert im August 1942 geschrieben hatte.[32] Mauchly entwickelte hier die Vorteile eines „electronic computor“ gegenüber den damals üblichen mechanischen Techniken, insbesondere dem „mechanical analyser“, für das Ausführen von Berechnungen (Mauchly 1942, S. 355):

> „It is the purpose of this discussion to consider the speed of calculation and the advantages which may be obtained by the use of electronic circuits which are interconnected in such a way as to perform a number of multiplications, additions, substractions or divisions in sequence, and which can therefore be used for the solution of difference equations.“

Den Hauptvorteil eines „electronic computor“ sah Mauchly – neben der größeren Genauigkeit der errechneten Ergebnisse, einem leichteren und effizienteren Umgang mit bei der Berechnung auftretenden Fehlern, der einfacheren Umrüstung eines elektronischen Geräts für ein neues Problem und einer geringeren Anzahl von für den Betrieb benötigten Personen – in seiner großen Rechengeschwindigkeit. So ergab sich aus seiner Analyse der für die Lösung einer „numerical difference equation“ benötigten Zeit (Mauchly 1942, S. 358):

> „It is obvious, however, that by the use of even 10,000 steps at the speed of the electronic device [100,000 pulses per second], the time for the over-all integration is still extremely short – 100 seconds, in comparison to the time required for an equivalent manual integration – *at least* several hours – and also quite short in comparison to the time required for solution with less accuracy on a mechanical differential analyser (15 to 30 minutes).“

Nachdem der noch vage Vorschlag für einen „electronic computor“ weiter ausgearbeitet war, wurden zunächst 150.000 US-Dollar vom Militär zur Verfügung gestellt, ein Budget, das bis zur Fertigstellung auf 460.804 US-Dollar aufgestockt wurde (vgl.

---

31 Der „differential analyser“ war ein Analogrechner, den Vannevar Bush bereits vor dem zweiten Weltkrieg entworfen und gebaut hatte.

32 Mauchly kannte zu dieser Zeit die Arbeiten von Atanasoff, der bereits 1935 am Iowa State College begonnen hatte, einen digitalen, elektronischen Computer zu bauen. Während die elektronischen Teile von Atanasoffs Maschine fertiggestellt werden konnten, wurde ein „binary card reader“ – unabdingbarer Bestandteil der Maschine – nie wirklich funktionstüchtig. Das Projekt wurde deshalb 1942, als Atanasoff Iowa verließ, eingestellt (vgl. Randell 1982, S. 294).

Williams 1985, S. 276, 302). Das ENIAC-Projekt, welches das mit Abstand komplexeste elektronische Gerät seiner Zeit vorsah, wurde von Ingenieuren sowohl der Armee als auch der Moore School heftig als undurchführbar kritisiert. Es ist deshalb äußerst unwahrscheinlich, daß diese Gelder auch ohne den drängenden militärischen Rechenbedarf, den der Zweite Weltkrieg mit sich brachte, zur Verfügung gestellt worden wären. Der zweite Weltkrieg war also für die Entwicklung des ENIAC zumindest zu diesem Zeitpunkt wichtige Voraussetzung (Heintz 1993, S. 214): „Ohne staatliche Gelder hätten diese kostspieligen [elektronischen] Computerprojekte niemals durchgeführt werden können".

Am 31. 5. 43 begann die tatsächliche Arbeit am ENIAC. Da die „Moore School" um 1940 ein Zentrum für die Konstruktion elektronischer Geräte war, die für die Entwicklung des Radars gebraucht wurden, konnte bei der Entwicklung des ENIAC auf diese Erfahrungen zurückgegriffen werden, zumal Eckert und auch die meisten anderen Mitglieder des Entwicklungsteams bereits über einen großen Erfahrungsschatz mit solchen Radarprojekten verfügten.[33] Mauchly hatte zudem bereits einen analogen Computer, einen „harmonic analyser", und einen „digital computer of a very special kind, for ciphers which used the storage features of neon or gas tubes" (Randell 1982, S. 297) gebaut.

Während Mauchly für das konzeptionelle Design zuständig war, bestand die Aufgabe Eckerts in dem Entwurf der einzelnen Schaltkreise. Bestimmend war dabei für beide eine pragmatische Herangehensweise (Williams 1985, S. 275f.):

> „Eckert was insistent that each part of the machine be designed in a simple, straightforward manner. The whole thrust was to get the project moving. If the choice had to be made between an elegant engineering solution to a problem and a brute force, ugly solution which was known to work, the choice was invariably the latter."

Dem Anliegen entsprechend, daß die Maschine baldmöglichst funktionieren sollte, waren viele der elektronischen Schaltkreise, die die grundlegenden arithmetischen Berechnungen ausführten, elektronische Kopien der entsprechenden Einheiten, die in damals üblichen mechanischen Rechengeräten verwendet wurden – ein Vorgehen, das sich schon 1942 in Mauchlys Text andeutete (Mauchly 1942, S. 356):

> „As already stated, the electronic computor utilizes the principle of counting to achieve its results. It is then in every sense the electrical analogue of the mechanical adding, multiplying and dividing machines which are now manufactured for ordinary arithmetic purposes."

---

33 Durch den Krieg wurde Elektronik allgemein zu einem entscheidenden Rüstungsgut. Während die Elektronikindustrie in den USA vor dem Krieg vorwiegend mit der Produktion von Radiogeräten beschäftigt war, mußten nun Funkgeräte, Radar, Flugzeug- und Steuerungselektronik in ständig wachsenden Größenordnungen bereitgestellt werden. Allein die Entwicklung des Radars verschlang dabei 2,5 Milliarden US-Dollar, die Miniaturisierung elektronischer Bauteile entsprechend den militärischen Anforderungen 1 Milliarde US-Dollar (vgl. dazu Noble 1986a). Das dabei entstehende „know-how" war eine wesentliche Voraussetzung für den Bau elektronischer Computer.

Wohl nicht zuletzt auf Grund des pragmatischen Vorgehens konnte das ENIAC–Projekt schließlich, allerdings erst nach Beendigung des Krieges, erfolgreich abgeschlossen werden. Der ENIAC war damit der erste funktionsfähige elektronische Digitalcomputer. Er enthielt 1500 elektromechanische Relais und 18000 Vakuumröhren und füllte einen ganzen Saal. Wie es Mauchly 1942 in Aussicht gestellt hatte, war er erheblich schneller als die anderen – mechanischen oder elektromechanischen – Rechenmaschinen seiner Zeit.[34]

Die Programmierung des ENIAC erfolgte durch das manuelle Setzen mechanischer Schalter und das Stecken von Kabelverbindungen. 1942 hatte Mauchly diese Art der Programmierung noch als sehr komfortabel beschrieben. Als einen Vorteil einer elektronischen Rechenmaschine benannte er (Mauchly 1942, S. 355):

> „Thirdly, the ease with which the various components of such a computing device can be interconnected by cables and switching units makes it possible to set up a new problem without much difficulty."

Für den ENIAC waren damit keine Vorkehrungen getroffen worden, „for setting up a program automatically", wie im „ENIAC Progress Report" vom 31. 12. 43 festgestellt wird, (vgl. Williams 1985, S. 301) und wie das z. B. beim Einlesen eines Programms von Lochkartenstapeln der Fall ist. Dies hätte zu unnötigen Verkomplizierungen des Maschinenentwurfs geführt, zumal der ENIAC vor allem für ballistische Berechnungen vorgesehen war und infolgedessen das Programm nicht allzu oft geändert werden mußte. Eine zeitaufwendige Programmierprozedur war von daher vertretbar.

Im Januar 1944 war das grundlegende Design des ENIACs abgeschlossen, so daß mit dem Bau der Anlage begonnen werden konnte. Im Juli 1944 waren zwei Akkumulatoren und einige andere wichtige Bauteile funktionstüchtig. Von Ende 1943 an hatten Eckert und Mauchly deshalb mehr Zeit, sich mit über den ENIAC hinausgehenden Problemen zu beschäftigen. Ein solches Problem betraf die nun im Gegensatz zur hohen Rechengeschwindigkeit doch als langwierig empfundene Programmierung des ENIAC, an die Arthur W. Burks, ein Mitglied der Entwicklungsgruppe, sich im Nachhinein wie folgt erinnert (Burks 1987a, S. 4):

> „On the ENIAC, this programming was done by hand: by setting mechanical switches of the program controls of each of the computing units used in the problem, interconnecting these program controls with cables, and setting the switches of the function tables. The procedure was long, laborious, and difficult to check; moreover, while it was done, the machine stood idle."

Nach entsprechenden Diskussionen mit Mauchly schrieb Eckert im Januar 1944 ein Papier[35], das sich mit dem Problem der zeitaufwendigen Programmierung des ENIAC

---

34 Für eine kurze technische Beschreibung des ENIAC siehe (Burks 1987a). Eine detaillierte Darstellung ist (Goldstine & Goldstine 1946).

35 Dieses ist in (Eckert 1980) unter dem Titel „Disclosure of a Magnetic Calculating Machine" abgedruckt.

auseinandersetzte. Die Programmierung sollte nicht mehr durch das Stecken von Kabeln, also gerätetechnischen Manipulationen von außen, erfolgen. Statt dessen sollte ein Programm in einem elektronischen Speicher abgelegt und genau wie die zu berechnenden Zahlen physikalisch durch elektronische Schaltelemente repräsentiert werden. Das Einlesen von Programmen konnte so (z. B. mit Hilfe eines Lochkartenlesers) mit großer Geschwindigkeit erfolgen. Der Programmiervorgang an der Maschine wurde erheblich beschleunigt. Dadurch wurde eine bessere Auslastung der Maschine gewährleistet, die jetzt nicht mehr während der zuvor langen Umprogrammierungszeit stillstehen mußte.

Die Idee einer „magnetic calculating machine", die in Eckerts 44er-Papier dargestellt ist, wurde wenig später aufgegeben zugunsten der Idee, für den für die Speicherprogrammierung benötigten großen und schnellen Speicher „mercury delay lines" zu verwenden. Solche Quecksilberröhren waren ursprünglich für die Radarausrüstung des Zweiten Weltkrieges entwickelt worden und wurden nun von Eckert und T. Kite Sharpless so modifiziert, daß 1024 zirkulierende Impulse mit Hilfe einer solchen Röhre und einigen Vakuumröhren gespeichert werden konnten.[36] Die neue Speichertechnologie erlaubte, den EDVAC ganz anders zu konstruieren als den ENIAC. Insbesondere konnte auf parallele Verarbeitungsprozesse verzichtet werden (Burks 1987a, S. 5f.):

> „Because the EDVAC would be so much faster, smaller and simpler than the ENIAC, the use of parallelism to gain speed was no longer needed and so it was decided to store numbers serially and process them serially. The guiding principle of EDVAC design was: One thing at a time, down to the last bit! All of this made it possible for the EDVAC to be much more powerful than the ENIAC and yet much smaller, perhaps with only 3000 vacuum tubes."

Elektronische Verarbeitung und Speicherung, zusammen mit den Daten gespeichertes Programm, Dualdarstellung und sequentielle Befehlsverarbeitung – Eigenschaften, die heute als charakteristisch für die Von-Neumann-Architektur angesehen werden, – waren also bereits diskutiert und in Planung, bevor von Neumann im Sommer 1944 zu der Gruppe stieß.[37]

---

36 Dazu wurde an beiden Enden der Quecksilberröhre je ein Quarzkristall angeschlossen, der aus elektrischen Impulsen physikalische bzw. akustische Signale erzeugen konnte bzw. umgekehrt. Ein elektrischer Impuls auf der Eingangsseite wurde von dem ersten Quarzkristall in ein akustisches Signal umgewandelt und als solches, durch das Quecksilber erheblich verlangsamt, durch die Röhre geschickt. Auf der anderen Seite angelangt wurde es durch den zweiten Quarzkristall wieder in ein elektrisches Signal umgewandelt und an den Eingang der Röhre zurückgesendet. So konnten zirkulierende Signale gespeichert werden (vgl. Burks 1987a, S. 5).

37 Entsprechend erinnert sich Burks, Mitglied der ursprünglichen ENIAC-Gruppe (Burks 1987b, S. xv): „The Moore School staff had begun to consider the prospects of storing instructions in the machine at least eight months before von Neumann's involvement with the project, and the ideas that went into the report [First Draft] were dicussed in great detail by Eckert, Mauchly, Arthur Burks, Goldstine and von Neumann before the writing of the report." Und Bauer (1998, S. 85) stellt fest: „Es gibt keinen Grund daran zu zweifeln, daß Eckert und Mauchly das, was später funktionell unter der ‚von-Neumann-Maschine' verstanden wurde, ausgetüftelt hatten,

Bei einem Beratungsbesuch im „Ballistic Research Laboratory“ des „Aberdeen Proving Ground“ im Sommer 1944 wurde von Neumann auf dem Bahnhof von Herman Goldstine, der zufällig auch dort wartete, angesprochen (vgl. Goldstine 1972, S. 182). Als im Verlauf ihrer Unterhaltung Goldstine von dem ENIAC-Projekt erzählte, wurde von Neumann – auch in seiner Eigenschaft als Berater des Atombombenprojektes von Los Alamos, in dem hohe Rechenkapazität benötigt wurde – aufmerksam. Ab ungefähr August 1944 wurde von Neumann ein regelmäßiger Besucher des ENIAC-Projektes. Einer Schilderung Goldstines zufolge scheint ihn der Rechner damals sehr beeindruckt zu haben (Goldstine 1979, S. 54):

> „Als er die Maschine kennenlernte, freute er sich sehr darüber, und sie regte sein Interesse für das Gebiet weiter an. Die Maschine war auch in ihrer damaligen, unvollendeten Form genau das, was von Neumann zu seinen Rechnungen benötigte, die er bis dahin in Los Alamos so mühselig ausgeführt hatte.“

Das Los-Alamos-Projekt wurde zusammen mit dem Ballistic Research Laboratory schon mit Beginn der Testphase des ENIAC im Frühjahr 1945 der erste, intensivste und auch sehr zufriedene Nutzer, wie folgender Ausschnitt aus einem Brief des Leiters der Los-Alamos-Laboratorien an das ENIAC-Team zeigt (nach Lindner et al. 1988, S. 98):

> „Die Berechnungen, die bereits auf dem ENIAC abgeschlossen wurden, wie auch die gegenwärtig durchgeführten, sind außerordentlich wertvoll für uns [...]. Die Komplexität dieser Probleme ist so groß, daß es nahezu unmöglich gewesen wäre, eine andere Lösung ohne die Hilfe des ENIAC zu erhalten. Wir sind außerordentlich glücklich darüber, für diese genauen Berechnungen den ENIAC zur Verfügung zu haben.“

Doch von Neumanns Interesse galt nicht nur dem ENIAC. Auch für das bereits in Planung befindliche Vorhaben, einen speicherprogrammierbaren, elektronischen Rechner zu bauen, begeisterte er sich recht schnell. Das wird vielleicht am deutlichsten dadurch dokumentiert, daß erst auf Grund seines Einsatzes dem Projekt weitere 100.000 US-Dollar für die Entwicklung des EDVAC genannten Folgerechners vom „Army Ordonance Department“ bewilligt wurden (vgl. Williams 1985, S. 302). Von Neumann trug so durch seine Bekanntheit und seinen Einfluß dazu bei, die finanziellen Voraussetzungen für Planung und Bau des EDVAC zu schaffen. Der Rechenaufwand, der durch den Bau und die anschließende Vervollkommnung der Atomtechnik verursacht wurde, war so ein wichtiger treibender Faktor für die Entwicklung von Hochgeschwindigkeitsrechengeräten.

---

bevor von Neumann, initiiert durch Herman Goldstine, mit dem Aberdeen Proving Ground und der Moore School Kontakt hatte.“

2.2. Text: „First Draft of a Report on the EDVAC“

Ab dem Spätsommer[38] 1944 nahm von Neumann an Gesprächen der ENIAC-Gruppe teil, die sich nun um den Bau eines besseren Rechners drehten. Nachdem er im März und April 1945 drei Tage an der Moore School verbracht hatte und dort mit dem ENIAC-Team über Hardwaredesign, Speicherorganisation und die arithmetische Ausrüstung diskutiert hatte, schrieb er das Papier, das als „First Draft of a Report on the EDVAC“ bekannt wurde, und schickte es an Goldstine. Dieser ließ das Papier tippen und verteilte es breit an Interessierte.

Der „First Draft“ war das erste Papier, das detailliert das Konzept eines speicherprogrammierbaren, elektronischen Rechners einer breiten Fachöffentlichkeit vorstellte. Von Neumann beschrieb hier ein „Gerät“, das besteht aus den Komponenten „Central Arithmetic (CA)“, „Central Control (CC)“, „Memory (M)“, „Input (I)“, „Output (O)“ und einem „outside recording medium (R)“. Daten und Programme können in dem Gerät gleichermaßen im „Memory“ abgelegt werden. Das hier vorgestellte Konzept, später so oder in ähnlicher Form als „Von-Neumann-Architektur“ bekannt geworden,[39] wird heute in der Regel als grundlegend für den Aufbau der meisten modernen Computer angesehen. In der Wissenschafts-Community, die sich nach dem Krieg mit dem Bau von Rechenmaschinen befaßte, wurde es international rezipiert (Heintz 1993, S. 219):

> „Obwohl der Bericht erst viele Jahre später publiziert wurde, fand er informell rasche Verbreitung. Wer sich mit Computern beschäftigte, und das war zu dieser Zeit noch eine überschaubare Gruppe, deren Mitglieder sich praktisch alle persönlich kannten, hat den Bericht gelesen und sich an ihm orientiert.“[40]

Daß von Neumann als der einzige Autor dieses Papiers aufgeführt ist, erregte großen Ärger bei den Mitgliedern der ursprünglichen ENIAC-Gruppe. Besonders Eckert und Mauchly fühlten sich um ihren Anteil an den Ideen gebracht, zumal die Mitglieder der ENIAC-Gruppe im Gegensatz zu von Neumann der militärischen Geheimhaltungs-

---

38 Strittig ist, ob ab August oder September. Die Genehmigung der Sicherheitsbehörde, die Moore School besuchen zu dürfen, erhielt von Neumann erst am 7. September 1944, so daß Mauchly dies als den frühestmöglichen Zeitpunkt für einen Besuch ansieht. Goldstine vertritt jedoch die Auffassung, daß für von Neumann die Genehmigung nur eine Frage der Formalität war und er bereits am 7. August seinen ersten Besuch tätigte (vgl. Bauer 1998, S. 85).

39 Nach (Bauer 1998, S. 84) ist der „First Draft“ selbst das Kriterium, das festlegt, was ein Von-Neumann-Rechner ist: „Einigermaßen sicher ist nur, was mit einem Von-Neumann-Rechner gemeint ist: ein Rechner, der sich im funktionellen Aufbau an das Schriftstück *First Draft of a Report on the EDVAC* anlehnt, das, in Erfüllung des *Contract No. W-670-ORD-4926 between the United States Army Ordnance Department and the University of Pennsylvania, Moore School of Electrical Engineering*, auf den 30. Juni 1945 datiert ist und von John von Neumann abgezeichnet wurde.“

40 Ähnlich stellt (Burks 1987b, S. xv) zur Rezeption des „First Draft“ fest: „The first widely circulated document on the stored program computer, it served as a primer for subsequent stored-program computer design based on the serial delay line memory.“

pflicht unterlagen und entsprechend nicht selbst veröffentlichen durften. Tatsächlich war eine Folge der Verteilung des „First Draft", daß Eckert und Mauchly ihr Ansinnen auf eine Patentierung ihrer Ideen nicht durchsetzten konnten. Bei den entsprechenden Gerichtsverhandlungen im April 1947 wurde der „First Draft" als eine Publikation dieser Ideen angesehen, die über ein Jahr zurücklag, was nach US-amerikanischem Recht eine Patentierung unmöglich machte (vgl. Williams 1985, S. 348).

Während Eckert und Mauchly also kommerzielle Interessen hatten, agierte von Neumann eher als Wissenschaftler (Heintz 1993, S. 221):

> „Denn im Gegensatz zu anderen Computerpionieren, allen voran Presper J. Eckert und John W. Mauchly vom ENIAC-Team, hat John von Neumann seine Arbeit als wissenschaftlich verstanden und eine entsprechend offene Publikationspolitik verfolgt. ‚We are hardly interested in exclusive patents', schrieb er 1947 nach einer Sitzung, in der man die Patentprobleme rund um den ENIAC diskutiert hatte, ‚but rather in seeing that anything that we contributed to the subject, directly or indirectly, remains as accessible as possible to the general public.' (Zit. nach Aspray 1990, S.45) Um etwaige Patentansprüche zu verhindern, trugen alle IAS-Berichte den expliziten Vermerk, daß sie von öffentlichem Interesse seien."

Der „First Draft" spiegelt die wissenschaftliche Ausrichtung von Neumanns wider. Zwar zeigt er deutlich den Einfluß der technischen Diskussionen an der Moore School, doch ist er sehr viel theoretischer und allgemeiner gehalten als ein Konstruktionsplan einer speziellen Maschine, wie ihn andere Rechenmaschinenbauer der Zeit schrieben.[41] Burks urteilt entsprechend (Burks 1987b, S. xv):

> „However, it is also clear that von Neumann was an important catalyst and major contributor to these discussions [at the Moore School] and that the description of these ideas in a theoretical form, stripped of all engineering details, was his contribution alone. Much of the controversy that ensued unquestionably stems from differing perspectives of practicing scientists and engineers and from the value they attach to theoretical constructs, technical feasibility, engineering design, and construction."

Es war also besonders die theoretische Form, die erlaubte, von technischen Details abzusehen, die Burks als von Neumanns genuinen Anteil an dem „First Draft" sieht. Ähnlich beurteilten auch Eckert und Mauchly von Neumanns Beitrag kurz nach Erscheinen des „First Draft" und gestanden von Neumann zudem einen wesentlichen Anteil an der Gestaltung des Codes, einem ebenfalls eher theoretischen Arbeitsgebiet, zu (Eckert und Mauchly nach Davis 1995, S. 151):

> „He has contributed to many discussions on the logical control of the EDVAC, has prepared instruction codes, and has tested those proposed systems by writing out the coded instructions for specific problems. [...] In his report, the physical structures and devices [...] are replaced by idealized elements to avoid raising engineering problems which might distract attention from the logical considerations under discussion."

Zwar war der „First Draft" also Produkt der Diskussionen, die in der ENIAC-Gruppe geführt wurden. Die Art und Weise jedoch, die Perspektive, aus der von Neumann die Rechenmaschine beschrieb, unterschied ihn von der technisch-ingenieurwissen-

---

41 Das zeigt z. B. der Vergleich mit (Goldstine & Goldstine 1946), einem Text zum ENIAC, oder (Wilkes, Renwick 1950), einem Papier zum EDSAC.

schaftlichen Kultur der ENIAC-Gruppe. Um diese Perspektive genauer zu charakterisieren, werde ich zunächst Spuren der ingenieurwissenschaftlich-technischen Diskussionen in der ENIAC-Gruppe herausarbeiten, die sich in dem Entwurf der Rechenmaschine dokumentieren, so wie er im „First Draft" dargestellt ist, und anschließend auf Einflüsse aus der Neurophysiologie und der Mathematik eingehen. Abschließend betrachte ich die Konzeption des Codes.

Die folgenden Textanalysen lehnen sich an das Verfahren der Dokumentarischen Interpretation an, so wie ich es in Kapitel 1 genauer behandelt habe. Ich werde bei meiner Besprechung die in dem Text verwendete Begrifflichkeit soweit möglich beibehalten, was aus heutiger Sicht zuweilen befremdlich wirken kann. Da sich in dieser Zeit die Terminologie herausbildete, in der über Rechenmaschinen und ihre Programmierung nachgedacht wurde, bietet die Art der Benennung jedoch wertvolle Hinweise darauf, wie die neue Technik im Wissen der Zeit verortet und gedeutet wurde. Einige Textpassagen werde ich recht detailliert behandeln, was für den Versuch unerläßlich ist, den Orientierungsrahmen zu explizieren, vor dessen Hintergrund die Rechenmaschine und Programmierung verstanden werden. Zudem zeigt dieses „genaue Lesen", wie auch moderne informatische Fachtexte auf unterliegende Sinnzusammenhänge hin untersucht werden können.

2.2.1. Technischer Diskurs

Sein Ursprung in der ingenieurwissenschaftlich-technisch ausgerichteten Diskussion, die von Neumann, Eckert, Mauchly und die anderen Mitglieder der ENIAC-Gruppe geführt haben, ist dem „First Draft" deutlich eingeschrieben: Bei dem „Gerät", das von Neumann in dem „First Draft" beschreibt, scheint es sich zunächst um eine konkrete, physikalische, im wesentlichen aus Vakuumröhren bestehende Maschine zu handeln. Nach einer Aufzählung und kurzen Diskussion möglicher mechanischer und elektronischer Elemente für ein digitales Rechengerät und einem Exkurs über deren Ähnlichkeit mit menschlichen Neuronen stellt er auf Grund der hohen „reaction time" (10^{-6} seconds) von Vakuumröhren abschließend fest (von Neumann 1945, 4.3)[42]:

> „In the considerations which follow we will assume accordingly, that the device has vacuum tubes as elements. We will also try to make all estimates of numbers of tubes involved, timing, etc. on the basis, that the types of tubes used are the conventional and commercially available ones."

Er favorisiert also Vakuumröhren als Grundbestandteile seines Geräts und legt Wert darauf, seine Erwägungen auf tatsächlich im Handel erhältliche Röhren zu beziehen.

---

42 Ich zitiere den „First Draft" nach den detaillierten Gliederungsnummern statt nach Seitenzahlen, weil sie so in jeder der mittlerweile zahlreichen Veröffentlichungen leichter wiedergefunden werden können.

Es ging also darum, daß das Gerät auf Grundlage der verfügbaren technischen Mittel tatsächlich gebaut werden konnte. Entsprechend wird in dem „First Draft“ der technische Aufbau von Vakuumröhren und wie man diese ursprünglich analogen Elemente als digitale Bausteine verwenden kann diskutiert (vgl. 5.1). Außerdem sind immer wieder Überlegungen dazu eingefügt, wie der Entwurf mit Hilfe von Vakuumröhren realisiert werden kann, wieviele Vakuumröhren benötigt werden, wie diese kombiniert werden müssen und wo der Einsatz von Vakuumröhren an seine Grenzen stößt (vgl. bes. 6.5, 7.6 und die Passagen zum „memory“, 12.5ff.). Wie bei der Konstruktion von Maschinen üblich, spielen auch Kostenabwägungen in dem „First Draft“ eine Rolle. So bildeten Überlegungen zur Menge der benötigten Ausrüstung die Grundlage für die Entscheidung, daß das Gerät genau die Schaltkreise für die algebraischen Operationen „+“, „–“, „x“, „÷“, „•“ und nicht nur „+“ und „–“ oder zusätzliche Operationen wie den Logarithmus beinhalten sollte.

Technische Gegebenheiten bilden auch die Grundlage für weitere zentrale Designentscheidungen, die den generellen Aufbau und die Struktur des Geräts betreffen. So leitet von Neumann aus der kurzen „Reaktionszeit“ der damals erhältlichen Vakuumröhren sowie aus Anforderungen an die Zuverlässigkeit ab, daß die EDVAC sich – im Gegensatz zu vorher existierenden Rechengeräten, in denen eine Erhöhung der Rechengeschwindigkeit durch eine Verfielfachung der benötigten Ausrüstung erkauft würde – ein gegenteiliges Prinzip zunutze machen soll (5.5):

> „[...] the reaction time of a not too complicated vacuum tube device can be made as short as one microsecond. [...] this is so much faster than any conceivable non vacuum tube device, that it actually produces a serious problem of keeping the device balanced, that is to keep the necessary human supervision beyond its input and output ends in step with its operations.“

Von Neumann folgert einige Zeilen weiter (5.6):

> „The device should be as simple as possible, that is, contain as few elements as possible. This can be achieved by never performing two operations simultaneously, if this would cause a significant increase in the number of elements required. The result will be that the device will work more reliably and the vacuum tubes can be driven to shorter reaction times than otherwise.“

Auf Grund der hohen Verarbeitungsgeschwindigkeit der neuen elektronischen Bauelemente soll also nur eine Operation zu jedem Zeitpunkt durchgeführt werden – z. B. das Transferieren eines bits eines neuen Befehls oder eines bits einer Zahl aus dem Speicher in die ausführenden Einheiten.

Der nach dem „First Draft“ am schwierigsten zu realisierende Maschinenteil ist der Speicher, der für die vorgesehene Programmspeicherung für damalige Verhältnisse sowohl sehr groß als auch sehr schnell sein mußte (12.4):

> „It [the observation that it is desirable to have a memory with a capacity of about 250.000 units] shows in a most striking way where the real difficulty, the main bottleneck of an automatic very high speed computing device lies: At the memory.“

Um einen solchen Speicher zu bauen, ist es dem Papier zufolge nötig, auf „delay elements of very great efficiency“ (12.5) zurückzugreifen, ein Hinweis auf die von der

ENIAC-Gruppe bereits getestete „mercury delay line"-Technologie. Entsprechend diesem zyklischen Speicherverfahren, bei dem auf die einzelnen bits einer gespeicherten Zahl oder eines gespeicherten Befehls nur nacheinander zugegriffen werden kann, arbeitet auch der Speicher des Geräts seriell: Ein bit einer Zahl oder eines Befehls nach dem anderen wurde aus dem Speicher zu der entsprechenden verarbeitenden Einheit transferiert und umgekehrt sowie die Zahlen bit für bit addiert, subtrahiert usw. Die streng serielle Verarbeitung von Zahlen wie auch Befehlen, die im „First Draft" vorgeschlagen wird, erklärt sich also auch aus der für den Bau des EDVAC vorgesehenen Speichertechnologie.[43]

Auch für die Entscheidung, daß das Gerät binär arbeiten sollte, d. h. daß intern Zahlen und Programm im Dualsystem dargestellt werden, sind Überlegungen zum technischen Aufbau, hier von Vakuumröhren, zentral (5. 1):

> „Thus, wether the tubes are used as gates or as triggers, the all-or-none, two equilibrium arrangements are the simplest ones. Since these tube arrangements are to handle numbers by means of their digits, it is natural to use a system of arithmetics in which the digits are also two valued. This suggests the use of the binary system."

Weil also zwei stabile Gleichgewichtszustände mit Hilfe von Vakuumröhren technisch am einfachsten zu realisieren sind (einfacher als z. B. Arrangements von Vakuumröhren, die drei, vier oder mehr Gleichgewichtszustände aufweisen), wird im „First Draft" ein entsprechendes Zahlensystem, das binäre, favorisiert. Als weitere Gründe für diese Entscheidung gibt von Neumann an, daß auch menschliche Neuronen „all-or-none elements" sind (5.1) – deutet also an, daß auch das menschliche Gehirn letztlich binär arbeitet –, und betont schließlich die Einfachheit binärer gegenüber dezimaler Arithmetik (5.2). Die letzten beiden Gründe verweisen auf den konstitutiven Anteil, den auch neurophysiologische und mathematische Diskurse an dem „First Draft" hatten. Darauf werde ich in den nächsten beiden Abschnitten noch eingehen.

Von Neumanns Überlegungen und Modell knüpften also an die technischen Gegebenheiten in dem ENIAC-Projekt an. Wichtige Designentscheidungen – die hardwaretechnisch zu realisierenden algebraischen Operationen, sequentielle Daten- und Befehlsverarbeitung bis hinunter auf die Bitebene, Dualdarstellung – leitete er aus technischen Bedingungen und Möglichkeiten ab. Es ging ihm darum, ein Gerät zu beschreiben, das mit Hilfe verfügbarer Technik gebaut werden konnte, und in dieser Hinsicht erscheint das Gerät als reale, materielle Maschine. In dem „First Draft" dokumentiert sich also die ingenieurwissenschaftlich-technische Kultur des ENIAC-Projekts, in dem es vor allem um den baldmöglichsten Bau einer funktionierenden Maschine ging.

---

43 In (12.8) diskutierte von Neumann auch die Möglichkeit, eine für die parallele Verarbeitung geeignete Speichertechnolgie zu verwenden, hob aber hervor, daß auf Grund von Adressierungsproblemen der Speicher letztlich sequentiell organisiert sein müsse (vgl. meinen Abschnitt 2.4 zur IAS-Maschine).

2.2.2 Neurophysiologischer Diskurs

Zwar scheint es sich bei dem „Gerät" des „First Draft" zunächst um eine physikalische Maschine zu handeln, deren elementare Bestandteile Vakuumröhren sind. Doch schon in der einleitenden Definition „An automatic computing system is a [...] device which can carry out instructions to perform calculations" bleibt offen, ob das „automatic computing system" als Maschine gedacht wird. Im Vergleich mit zwei rein ingenieurwissenschaftlich-technischen Texten der Zeit, die im Rahmen des ENIAC-Projektes entstanden – dem bereits erwähnten „The Use of High Speed Vaccuum Tube Devices for Calculating" (Mauchly 1942) und „The Electronic Numerical Integrator and Computer" (Goldstine and Goldstine 1946) – fällt auf, daß der „First Draft" auch für einen ingenieurwissenschaftlichen Text eher ungewöhnliche Gedankengänge entwickelt: Während die beiden genannten Papiere Vakuumröhren als die Grundbestandteile der beschriebenen bzw. geplanten Rechner vorstellen und deren Geschwindigkeit (bes. Mauchly) und den Aufbau (bes. Goldstine) beschreiben, führt von Neumann Vakuumröhren als nur eine mögliche Realisierung, einen Spezialfall digitaler Elemente allgemein ein (vgl. Abschnitt 4 mit dem Titel „Elements, Synchronism Neuron Analogy"), zu denen er neben mechanischen und elektronischen Bauteilen auch menschliche Neuronen zählt: „4.2 It is worth mentioning, that the neurons of the higher animals are definitely elements in the above sense [relay-like elements, with discrete equilibria]". Da Vakuumröhren von all diesen Elementen die kürzeste „reaction time" besäßen, kürzer auch als das menschliche Neuron, wie er anmerkt, favorisiert er Vakuumröhren als Grundbestandteile des Geräts (vgl. 4.3).

Wie die Diskussion menschlicher Neuronen neben technischen Bauteilen zeigt, setzt von Neumann sein Gerät in anderer Weise mit dem Menschen in Beziehung als Mauchly oder die Goldstines. Zwar führen auch Mauchly und die Goldstines Vergleiche ihrer Technik mit menschlichen Tätigkeiten durch: Mauchly errechnet, daß eine manuelle, im Vergleich zu einer mit einem „electronic computor" durchgeführte Integration mindestens mehrere Stunden im Gegensatz zu nur 100 Sekunden in Anspruch nähme (vgl. S. 358); und die Goldstines, die allerdings den „First Draft" zu dieser Zeit bereits kannten, vergleichen menschliches und maschinelles Rechnen (jedoch nur in diesem einen Satz) auch in bezug auf die Art, wie es durchgeführt wird (S. 369): „For the ENIAC, as for human computers, the computation must consist of a series of arithmetical operations." Von Neumann postuliert jedoch, daß elektronische Elemente prinzipiell genauso funktionieren wie menschliche Neuronen und unterstellt damit eine starke Ähnlichkeit zwischen dem Aufbau seines Geräts und dem des menschlichen Nervensystems und Gehirns.[44]

---

44 Der Vergleich zwischen der Funktionsweise des menschlichen Gehirns und elektronischen Schaltelementen war auch Atanasoff 1940 bereits eine Bemerkung wert (S. 319): „This circuit

Diese Idee, daß menschliches Nervensystem und Gerät sehr ähnlich sind, durchdringt die gesamte Terminologie des „First Draft": Das Gerät besitzt Organe – „specialized organs" für Addition, Subtraktion, Multiplikation, Division und Wurzelziehen (vgl. z. B. 2. 2), „general control organs" (vgl. z. B. 2. 3), „memory organs" (vgl. z. B. 7. 6, 12. 5) usw. Ihrerseits sind diese aus Unterorganen bzw. aus Netzwerken von „E-elements" aufgebaut, die als „analogs of human neurons" (6.2) einführt werden. Die Organe, Netzwerke und E-Elemente erhalten „Stimuli", sie sind „excited" oder „quiescent". Beispielsweise lautet die Definition und Beschreibung von E-Elementen (6.2, 6.4):

> „The element which we will discuss, to be called an E-element, will be represented by a circle, which receives the excitatory and inhibitory stimuli, and emits its own stimuli along a line attached to it [...]. An E-Element receives the stimuli of its antecedents across excitatory synapses [...] or inhibitory synapses."

Für die Beschreibung der „Gedächtnisorgane"[45] greift von Neumann auch auf das damals in der Neurophysiologie weit verbreitete Neuronenkreismodell zurück, nach dem die in einem Neuronenkreis zirkulierenden Signale den Schwellenwert[46] eines angegliederten Neurons senken und so Vorgänge wie Lernen und Konditionierung ermöglichen. Allerdings interpretiert von Neumann die Aufgabe solcher Neuronenkreise im Nervensystem neu, wie Katharina Schmitt-Brücken (1998) zeigt: Während zuvor das bloße Zirkulieren der Signale im Neuronenkreis die Senkung des Schwellenwertes eines angeschlossenen Neurons bewirkte und damit – so die Theorie – z. B. die Auslösung einer bestimmten Reaktion vereinfacht wurde, wurde nun wichtig, welche Signale in welcher Reihenfolge in dem Neuronenkreis zirkulierten. Die kreisenden Signale selbst waren bei von Neumann der gespeicherte Inhalt – eine Vorstellung, die sich im Anschluß auch für die Erklärung von Gedächtnisleistungen in der Psychologie durchsetzte. Gut zu sehen ist hier, wie sich Theorien über das menschliche Gedächtnis und der Rechnerbau gegenseitig beeinflußten. Hier liegt ein schönes Beispiel für den metaphorischen Zirkel vor, den ich in Abschnitt 1.2 beschrieben habe: zunächst wird die Neuronen-Terminologie für die Beschreibung technischer Bauteile verwendet; nach einer gewissen Zeit wird dann die technische Funktionalität dieser Bauteile auf die Erklärung neuronaler Prozesse rückübertragen.

Das Gerät des „First Draft" ist also aus hypothetischen Elementen aufgebaut, die als Analogon zu menschlichen Neuronen erscheinen und auch die Funktion von Vakuumrühren beschreiben können (6.1, 6.2):

---

operates upon new principles in the computing art, principles that are rather analogous to the function of the human brain in mental calculation." Allerdings blieb es bei ihm, im Gegensatz zu dem „First Draft", bei dieser einen Feststellung.

45 Von Neumann verwendet im „First Draft" durchgängig den Begriff „memory". Besonders in Texten aus England wurde hingegen der Begriff „store" gewählt.

46 Nur wenn die Anzahl der erregten Synapsen eines Neurons größer oder gleich dem sogenannten Schwellenwert ist, wird ein Neuron aktiviert.

„[...] we will base our considerations on a hypothetical element, which functions essentially like a vaccuum tube [...] but which can be discussed as an isolated entity, without going into detailed radio frequency electromagnetic considerations." „The analogs of human neurons [...] seem to provide elements of just the kind postulated at the end of 6.1."

Die E-Elemente standen dabei nicht in einer einfachen Eins-zu-eins-Beziehung zu Vakuumröhren (6.5):

„Comparison of some typical E-element networks with their vacuum tube realizations indicates, that it takes usually 1 – 2 vacuum tubes for each E-element. [...] On the average, however, counting two vacuum tubes per E-element would seem to be a reasonable estimate."

Das Gerät des „First Draft" ist also aus Neuronenmodellen zusammengesetzt. Entsprechend scheint auch das Gerät als gesamtes so aufgebaut zu sein wie das Nervensystem:

„2.6 The three specific parts CA, CC, (together C), and M corrrespond to the associative neurons in the human nervous system. It retains to discuss the equivalents of the sensory or afferent and the motor or efferent neurons. These are the input and the output organs of the device".

Die gewählte Begrifflichkeit bezieht sich also explizit auf den physikalischen Aufbau des Nervensystems, als dessen Teil auch das Gehirn betrachtet wird. Besonders deutlich wird dieser Rückbezug auf körperliche Gegebenheiten auch an den Stellen zur Ein- und Ausgabe des Gerätes. So sollen die Eingaben an das Gerät in einer Weise gegeben werden, die es fühlen bzw. spüren kann („which the device can sense" (1.2)) – und zwar mit Hilfe geeigneter „Organe" für den „Input" (vgl. 2.6); genauso sollen die Ausgaben des Geräts durch menschliche Organe mehr oder weniger direkt fühlbar sein („can be [...] sensed more or less directly by human organs" (2.6)). Das Fühlen der menschlichen Organe und das Fühlen des Geräts und seiner Organe erscheinen damit als gleichartig; dem Gerät werden körperlich-sensorische Eigenschaften zugeschrieben. Das Gerät des „First Draft" wird durch diese Begrifflichkeit biologisiert und als eine Art Organismus gezeichnet.

Exkurs: McCulloch/Pitts und Turing

Mit seiner Vorstellung von menschlichen Neuronen lehnte sich von Neumann an das Papier „A Logical Calculus of the Ideas Immanent in Nervous Activity" an, das 1943 in dem „Bulletin of Mathematical Biophysics" erschien. Warren McCulloch, Neurophysiologe und Direktor des „Laboratory for Basic Research" des „Department of Psychiatry" der Universität von Illinois, und Walter Pitts, ein achtzehnjähriger Mathematiker und Logiker,[47] beschreiben hier das Funktionieren des Nervensystems und Gehirns als ein Zusammenwirken von Neuronen in Netzen. Dabei gehen sie davon aus, daß Neuronen, ähnlich logischen Schaltelementen, genau zwei Zustände besitzen

---

47 Eine kurze Darstellung der Lebensläufe von McCulloch und Pitts gibt (McCorduck 1979, S. 72ff.).

können. Das war damals in der Neurophysiologie eine anerkannte, allerdings auch umstrittene Lehrmeinung, die sich mittlerweile jedoch als zu vereinfachend herausgestellt hat.[48] Auf Grund dieser Annahme schien es ihnen möglich, Ereignisse im Nervensystem mit Hilfe der propositionalen Logik zu beschreiben: „Because of the ‚all-or-none' character of nervous activity, neural events and the relations among them can be treated by means of propositional logic." – so ihr einleitender Satz. In dem Papier weisen die Autoren nach, daß jeder endliche logische Ausdruck mit Hilfe ihrer Neuronenmodelle dargestellt werden kann. Ihr Ergebnis beweist, daß so einfache Elemente wie die Pitts/McCulloch-Neuronenmodelle zusammengeschlossen in Netzwerken komplexe Vorgänge berechnen können. Abgesehen von den vorgenommenen Vereinfachungen der Pitts/McCulloch-Modelle gegenüber echten Neuronen schien damit das Gehirn im wesentlichen ein machtvolles logisches Berechnungsgerät zu sein. Entsprechend ziehen McCulloch und Pitts am Ende ihres Artikels weitreichende Schlußfolgerungen in bezug auf das menschliche Denken, wie z. B. die, daß „in psychology, introspective, behaviouristic or physiological, the fundamental relations are those of two-valued logic" (S. 25).

Wie er und später Pitts dazu kamen, das Gehirn mit Hilfe eines logischen Kalküls zu beschreiben, schilderte McCulloch 1948 im Rahmen der Diskussion zu von Neumanns Vortrag „The General and Logical Theory of Automata" auf dem Hixon Symposium, auf den ich in Kapitel 3 noch genauer eingehen werde. Sein Hauptinteresse, so McCulloch hier, sei es gewesen zu erklären, wie etwas – und das kann man seinen Worten entnehmen: Großartiges und Faszinierendes – wie Mathematik jemals entstehen konnte (S. 319):

> „I came, from a major interest in philosophy and mathematics, into psychology with the problem of how a thing like mathematics could ever arise – what sort of thing it was. For that reason I gradually shifted into psychology and thence, for the reason that I again and again failed to find the significant variables, was forced into neurophysiology. [...] it was not until I saw Turings paper that I began to get going the right way around, and with Pitts' help formulated the required logical calculus. What we thought we were doing (and I think we succeeded fairly well) was treating the brain as a Turing machine;"

In McCullochs Bemühen zu erklären, was Mathematik eigentlich ist und wie sie entstand, stellte also Alan Turings Arbeit „On Computable Numbers with an

---

48 So schreibt Barbara Becker über den Stand der neurophysiologischen Forschung (1992, S. 145): „Das Neuron wird von den Emergentisten [einer neurophysiologischen Schule] nicht länger mit einem bloßen Ein-/Aus-Schalter gleichgesetzt, da die biochemischen Vorgänge im synaptischen Spalt nach heutigen Erkenntnissen als mindestens ebenso relevant erscheinen wie die rein physikalische Informationsübertragung [...]. Die synaptischen Informationsüberträgersubstanzen, die sogenannten Transmitter, weisen dabei eine große biochemische Vielfalt auf. [...] ‚Diese Vielfalt, die einerseits chemisch bedingt und andererseits dem elektrischen Effekt der Neurotransmitter zu verdanken ist, schafft eine verschwenderische Signalkombinatorik - rechnerische Möglichkeiten, zu denen ein rein elektrisch arbeitendes Neuron nicht ohne weiteres in der Lage wäre.' (Changeux 1984, S. 118)".

Application to the Entscheidungsproblem" von 1936 einen Wendepunkt dar. Um diese Aussage zu verstehen, ist es nötig, kurz auf das Turingsche Papier einzugehen:

Ein zentrales Problem in Turings Papier ist die Frage, wie man berechenbare Zahlen von nicht berechenbaren abgrenzen kann. Problematisch ist diese Frage für Mathematiker besonders dadurch, daß es sich bei „Berechenbarkeit" um einen intuitiven Begriff handelt: „Berechenbar" sind alle Zahlen, die ein Mensch prinzipiell berechnen kann. Es galt also, diesen Begriff mathematisch-formal zu fassen, dabei aber gleichzeitig zu motivieren, daß der gewählte Formalismus tatsächlich genau die Zahlen erfassen kann, die ein Mensch berechnen kann. Dazu nahm Turing in einem ersten Schritt an, daß die berechenbaren Zahlen genau diejenigen sind, die mit Hilfe endlicher Mittel errechnet werden können. Um dieses „endliche Mittel" in einem zweiten Schritt konkreter zu fassen, dachte er sich fiktive Maschinen aus. Die endlichen Mittel wurden nun durch diese Maschine als Kriterium für Berechenbarkeit konkretisiert. Turing konstatierte, daß genau dann, wenn eine Zahl (durch einen Menschen) berechenbar ist, sie sich von einer solchen Maschine niederschreiben läßt. In einem dritten Schritt galt es nun, diese Behauptung zu beweisen, wobei ein Beweis im mathematischen Sinne hier lediglich eine Plausibilisierung sein kann, da „berechenbar" ja ein intuitiver Begriff ist. Turings Plausibilisierung bestand darin, den Leser davon zu überzeugen, daß ein rechnender Mensch im Grunde auch nichts anders tut als seine Maschinen, damit also auch nicht mehr Zahlen berechnen kann als dies seine Maschinen können.

Turings Problem war damit – zumindest was das Rechnen als ein Teilgebiet der Mathematik angeht – McCullochs Frage nach dem Wesen und der Entstehung der Mathematik verwandt: Um den Berechenbarkeitsbegriff formal fassen zu können, mußte Turing sich auch mit der Frage auseinandersetzen, wie Menschen Zahlen berechnen. Seine Antwort bestand darin, Rechnen als Folge diskreter, elementarer Ziffernoperationen zu konzipieren: Seine Maschinen arbeiten auf einem Band, das in Felder unterteilt ist, die je ein Zeichen beinhalten können. Die Maschine kann Schritt für Schritt diese Felder abtasten und in Abhängigkeit von dem gelesenen Zeichen und einem inneren Zustand das Zeichen löschen, eines schreiben und das Band so bewegen, daß eines der beiden benachbarten Felder bearbeitet werden kann.[49]

McCulloch und Pitts nun weiteten diese Sicht des Rechnens auf das Denken insgesamt aus. Das Gehirn zeichneten sie als verknüpfte Struktur von Neuronenmo-

---

49 (Heintz 1993, Kapitel 4) stellt Turings Konzept unter der Kapitelüberschrift „Das Fließband im Kopf" in den breiteren Zusammenhang der Rationalisierungsbewegungen der Zeit. Wichtiges Element war hier die Zergliederung von Arbeitsabläufen in einfache, reglementierte Tätigkeiten (S. 163): „Die Formalisierung von Arbeitsabläufen setzt die Aufspaltung von komplexen Handlungsabläufen in elementare Operationen voraus, die dann anschließend nach Effizienzkriterien resynthetisiert werden. Damit werden Handlungen reproduzierbar und die Handelnden im Prinzip substituierbar, entweder durch Menschen oder durch Maschinen."

dellen, die in einem bestimmten Zeittakt elementare Operationen ausführten. Zwischen der Ankunft von Reizen auf eingehenden Synapsen und der Aktivität des Neurons lag nach ihnen eine Verzögerungszeit, der „synaptic delay" (McCulloch, Pitts 1943, S. 18): „Between the arrival of impulses upon a neuron an its own propagated impulse there is a synaptic delay of more than half a millisecond." Nach dieser Verzögerung wurde das Neuron in Abhängigkeit von den Reizen auf den eingehenden Synapsen und seinem Schwellenwert aktiv oder nicht.[50] McCulloch und Pitts zeigten also, wie man ein Bild des Rechnens, nach dem dies in diskrete, elementare Operationen unterteilbar ist, in ein Modell des Gehirns und Nervensystems umsetzen kann, nach dem dieses aus Elementen aufgebaut ist, die solche elementaren Operationen durchführen. So gesehen wurde das Gehirn mit Hilfe eines logischen Kalküls faßbar. In dem 43er-Papier schrieben Pitts und er (S. 24f.):

> „It is easily shown: first, that every net, if furnished with a tape, scanners connected to afferents, and suitable efferents to perform the necessary motor-operations, can compute only such numbers as can a Turing machine; second, that each of the latter numbers can be computed by such a net; [...] This is of interest as affording a psychological justification of the Turing definition of computability [...]: If any number can be computed by an organism, it is computable by these definitions, and conversely."

Zwar bezieht sich diese Textstelle auf den Vorgang des Berechnens einer Zahl. Jedoch hielten McCulloch und Pitts auch das Gehirn als gesamtes für das Äquivalent eines solchen Netzes und damit auf Grund der von ihnen postulierten Äquivalenz zwischen ihren Netzen und Turingmaschinen für das Äquivalent einer Turingmaschine. So hielten sie einige Zeilen später fest (S. 25): „To psychology, however defined, specification of the net would contribute all that could be achieved in that field". Denn wenn das Netz, welches das Gehirn realisiert, einmal spezifiziert ist, sind in dieser Sichtweise alle weiteren Fragen mit Hilfe der Mathematik und Logik beantwortbar.

Turing lieferte McCulloch die „signifikanten Variablen", mit deren Hilfe er die Entstehung und das Wesen der Mathematik erklären konnte; sein entstandenes Modell wandte McCulloch auf psychologische Prozesse insgesamt an. Das verweist darauf, daß mathematisches Denken und auch Denken allgemein für McCulloch im wesentlichen Rechnen – gedacht als maschineller Vorgang – war. Daß der Mensch überhaupt für ihn eine Maschine war, darüber ließ McCulloch 1948 in der bereits erwähnten Diskussion auf dem Hixon Symposium keinen Zweifel. Er antwortete auf von

---

50 Zm besseren Verständnis ein Beispiel (aus der Einleitung zu (McCulloch, Pitts 1943), S. 15): „As an example, we have a simple unit with excitatory inputs, *a* and *b*, and with a threshold of one. Synapses connected to active cells contribute one unit. Suppose at the start of the *i*th time quantum, both *a* and *b* are inactive. Then, at the start of the $(i+1)$th time quantum, the unit is inactive since the sum of two inactive synapses is zero. If *a* is active but *b* is inactive, the state of the unit at the start of the $(i+1)$th time quantum is active since the inactive synapse plus the active synapse is one, which equals the threshold. [...] This threshold-one unit with two excitatory inputs is performing the logical operation INCLUSIVE OR on its inputs, and becomes active only if *a* OR *b* OR BOTH *a* AND *b* is active."

Neumann: „Unfortunately for us in the biological sciences – or at least in psychiatry – we are presented with an alien, or enemy's machine.“ Um diese Maschine, von der wir nur wüßten, daß sie falsche Antworten produziere, verstehen zu können, suche er, so McCulloch, nach einer angemessenen Theorie. Seine und Pitts Theorie sollte dafür einen Einstieg bieten (vgl. Diskussion zu (von Neumann 1948), S. 319).

Doch zurück zu von Neumann: Zwar war von Neumann bewußt, daß Neuronen tatsächlich komplizierter funktionierten als Vakuumröhren. Angelehnt an das Papier von McCulloch und Pitts abstrahiert er im „First Draft“ jedoch von diesen Differenzen:

> „Following W. Pitts and W. S. MacCulloch [...] we ignore the more complicated aspects of neuron functioning: Thresholds, temporal summation, relative inhibition, changes of thresholds by after effects of stimulation beyond the synaptic delay etc.“ (4.2) „Since we have a strict synchronism of stimuli arriving only at times which are integer multiples of t, we may disregard phenomena of tiring, facilitation, etc. We also disregard relative inhibition, temporal summation of stimuli, changes of threshold, changing synapses etc. In all this we are following the procedure of W. Pitts and W. J. MacCulloch“ (6.4).

Obwohl er für die „E-Elemente“, aus denen er sein Gerät aufbaut, weitere Vereinfachungen vornimmt, z. B. nur Elemente mit Schwellenwert eins, zwei oder drei zuläßt (vgl. 6.4), bezeichnet er sie immer wieder als „analogs of human neurons“ (z. B. 5.1, 6.2). Die Analogie zwischen Neuronen und elektrischen Schaltelementen war ihm anscheinend so einsichtig, daß er sie nicht grundsätzlich in Frage stellte. Er übernahm seine Neuronenmodelle aus einem Papier, welches das Gehirn und Nervensystem zu einem logischen Rechengerät erklärt; seine Arbeit stellt sozusagen den Umkehrschluß dar: Wenn Gehirn und Nervensystem im wesentlichen ein logisches Rechengerät sind, dann muß man Rechengeräte auch mit Hilfe von neuronengleichen Elementen beschreiben können. Wie das Nervensystem Teil eines Organismus ist, wird auch das Gerät als Organismus beschreibbar, der sensorische und motorische Eigenschaften besitzt.

2.2.3 Mathematischer Diskurs

Während der „First Draft“ auf technischen und auch neurophysiologischen Diskussionen aufsetzt, präsentiert er sich auch als in mathematischen Traditionen begründet.[51] Schon die äußere Form weist darauf hin, daß das Papier der Feder eines Mathematikers

---

51 Das „computing system“, das von Neumann beschrieb, sollte „vor allem ein Rechner“ (vgl. 2.2) sein, und die erste Einheit, die von Neumann behandelte, war die „Central Arithmetic“. Ausgeführt werden sollten numerische Berechnungen, was auf eine andere wichtige mathematische Tradition verweist, die ihre Spuren im „First Draft“ hinterlassen hat, auf die ich hier jedoch nicht näher eingehe. Die Einführung digitaler Rechenmaschinen ging einher mit der Verbesserung und Ausarbeitung numerischer Methoden und Verfahren, ein Prozeß, an dem von Neumann nicht unerheblich beteiligt war.

entstammt. Von Neumann unterteilte das Papier in kleine, von 1.0 bis 15.5 durchnumerierte Abschnitte, was an den Aufbau mathematischer Texte, die aus Definition, Satz, Beweis zusammengesetzt sind, erinnert. Und wie von einem mathematischen Text erwartet, findet sich am Anfang eine Definition. Der zentale Gegenstand des Papiers – das automatische Rechensystem – wird hier durch seine Funktionalität bestimmt:

> „1.2 An automatic computing system is a (usually highly composite) device, which can carry out instructions to perform calculations of a considerable order of complexity".

In diesem per Definition Setzen dokumentiert sich der Habitus eines Mathematikers Hilbertscher Schule, als der von Neumann gesehen werden kann: Bis 1931, dem Jahr von Gödels Unvollständigkeitsbeweis, hatte sich von Neumann intensiv mit Hilberts beweistheoretischem Programm auseinandergesetzt.[52] Im Stil des „First Draft" drückt sich diese mathematische Sozialisation aus. So beschreibt Mehrtens eine der zentralen Denkformen der Hilbertschen Axiomatik wie folgt (Mehrtens 1990, S. 124):

> „Hilberts Axiomatik ist gerade solche Schöpfung, die tut, als sei sie Schöpfung aus dem Nichts. Die Zeichen, Namen, Regeln, alles zugleich wird geschaffen als eine Welt, über die der Schöpfer Herr ist."

Von Neumann macht sich im „First Draft" zur Kreation des „automatic computing system" diese demiurgische Attitüde zu eigen. Auf diese Weise eingeführt wirkt das automatische Rechensystem wie ein geschichtsloses Objekt. Der Gegenstand des Papiers erscheint nicht als gewordene, gebaute und vergängliche Maschine, sondern erhält zeitlosen Charakter. Das bewirkt zum einen, daß der Entstehungskontext des EDVAC-Projektes und damit des „First Draft" – die Arbeit der ENIAC-Gruppe, die Konstruktion des ENIAC, die daraus hervorgegangenen Pläne für den EDVAC und die in diesem Rahmen getroffenen Entwurfsentscheidungen – nicht thematisiert zu werden braucht. Das mag seinen Teil zu den auf den „First Draft" folgenden Streitigkeiten zwischen Eckert, Mauchly und von Neumann, die ich zu Beginn des Abschnitts 2.2 geschildert habe, beigetragen haben. Zum anderen ist – und das ist im Kontext dieser Untersuchung der interessante Punkt – damit die Voraussetzung gegeben, die Struktur des Geräts allein aus der in der Definition angegebenen Funktionalität abzuleiten. Diese Struktur erhält so allgemeingültigen – alle Systeme, die Instruktionen zur Durchführung komplexer Berechnungen ausführen, betreffenden – Charakter, worauf ich nun genauer eingehe.

Der Schöpfung des Gegenstands folgt im „First Draft" eine Analyse von dessen Funktionieren, also des Ausführens von Instruktionen zur Durchführung von komplexen Berechnungen: „2.1 In analyzing the functioning of the contemplated device, certain classificatory distinctions suggest themselves immediately." In 2.2 bis

---

52 (Burks 1987b, S. xiii): „Like many young European mathematicians, he was drawn into David Hilbert's metamathematical program. When this program foundered over Gödel's 1931 incompleteness results, his interest in logic languished."

2.5 wird anschließend die Grobaufteilung des Geräts in CA, CC und M im wesentlichen aus der Funktionalität des Geräts abgeleitet:

- Daß das Gerät „vor allem ein Rechner" sein soll, ist die Begründung für CA, die „Central Arithmetic" (vgl. 2.2);
- Daß das Gerät „all purpose" sein soll, begründet CC, die „Central Control". Denn dies erfordere, so von Neumann hier, daß zwischen speziellen Instruktionen für ein spezifisches Problem und generellen Organen für deren Ausführung (CC) unterschieden werden müsse. Die Eigenschaft des Geräts, Instruktionen ausführen zu können, war demnach der Grund für die Einführung von CC (vgl. 2.3);
- Daß es komplizierte Sequenzen von Operationen (speziell Berechnungen) ausführen soll, ist die Begründung für M, den „Memory" (vgl. 2.4).

Zentrale Designentscheidungen – der grundsätzliche Aufbau des Geräts – ergeben sich also scheinbar selbstverständlich aus der in der Definition des Gegenstands gesetzten Funktionalität – dem Ausführen von Instruktionen zur Durchführung komplexer Berechnungen. Eine an mathematische Texte erinnernde Argumentationsweise – das Ableiten von Folgerungen aus gegebenen Definitionen – erlaubt, von den konkreten technischen Gegebenheiten zu abstrahieren. Der Aufbau des Geräts erscheint so als allgemeingültige, allen Rechensystemen, die Instruktionen ausführen, unterliegende Struktur.

2.2.4 Das Rechengerät: Maschine – Organismus – naturgegebene Struktur

Der „First Draft" läßt sich lesen als ein Dokument, das zwischen drei Wissenschaftskulturen angesiedelt ist – der ingenieurwissenschaftlich-technischen, der neurophysiologischen und der mathematischen. In welcher Weise er sich als ein ingenieurwissenschaftlich-technischer Text präsentiert, habe ich in Abschnitt 2.2.1 aufgezeigt. Immer wieder werden Fragen behandelt, welche die technische Umsetzung der dargestellten Überlegungen betreffen; zudem werden aus technischen Gegebenheiten – wie der Verarbeitungsgeschwindigkeit von Vakuumröhren – wichtige Designentscheidungen abgeleitet, z. B. die streng sequentielle Daten- und Befehlsverarbeitung. Das Gerät des „First Draft" erscheint in dieser Hinsicht als eine elektronische Rechenmaschine, deren Aufbau durch die vorgesehenen Bauelemente wesentlich mitbestimmt wird, und dessen spezifische Materialität auch für prinzipiellere, die generelle Architektur betreffende Überlegungen eine Rolle spielt.

Schon daß das Kürzel „EDVAC" in dem „First Draft" nur einmal – nämlich in der Überschrift – erscheint, kann als Hinweis darauf gesehen werden, daß es nicht nur um

die Entwicklung dieser einen Maschine geht[53]. Der „First Draft“ entwickelt allgemeinere Prinzipien, als von dem Konstruktionsplan einer Maschine zu erwarten. Dafür ist der Rekurs auf die Neurophysiologie zentral, wie ich ihn in Abschnitt 2.2.2 dargestellt habe . Er ermöglicht, daß von Neumann in dem „First Draft“ nicht nur eine spezielle Maschine, den EDVAC, behandelt, sondern auf einer allgemeineren, die Technik transzendierenden Ebene diskutiert[54]. Denn das Gerät des „First Draft“ ist aus ideellen Elementen zusammengesetzt, die nach von Neumann sowohl menschliche Neuronen als auch Vakuumröhren modellieren können. Die materielle Umsetzung kann als elektronisch sowie auch als biologisch vorgestellt werden. Dieser materiellen Beliebigkeit entspricht, daß sich der generelle Aufbau des Geräts scheinbar selbstverständlich aus seiner Funktion ergibt, was ich in Abschnitt 2.2.3 als typische Argumentationsform eines Mathematikers Hilbertscher Schule herausgearbeitet habe. Das Gerät erscheint – zusammengesetzt aus hypothetischen Elementen – als eine Struktur, die der Funktion „Instruktionen ausführen, um komplexe Berechnungen durchzuführen“ (was auch organische rechnende Systeme – Menschen – können), allgemein unterliegt. In dem Oszillieren des Blicks – dem Zusammenspiel ingenieurwissenschaftlicher, neurophysiologischer und mathematischer Diskurse – dokumentiert sich die Hybridobjekt-Sichtweise: Das Gerät erscheint einmal als Maschine, dann als Organismus und schließlich als immaterielle Struktur, die biologischen wie technischen Umsetzungen der Funktion „Instruktionen ausführen, um komplexe Berechnungen durchzuführen“ allgemein unterliegt.

Die später sog. „Von-Neumann-Architektur” wurde entsprechend dem sich im „First Draft“ dokumentierenden Allgemeingültigkeitsanspruch nicht selten als naturgegeben rezipiert, so z. B. von Zemanek bei einem Vortrag über architektonische Leitideen (Zemanek 1994, S. 56):

> „Denn die Gliederung der Rechenmaschine, wie sie Konrad Zuse selbstverständlich vorhin beschrieben hat, nämlich Rechenwerk, Leitwerk und Speicherwerk, die ist ja naturgegeben. Da hat man den Herrn John von Neumann überhaupt nicht gebraucht dazu. Und hinterher wurde dann dieses Etikett der ‚John-von-Neumann-Architektur' aufgeklebt.”

Auch für Bauer ist zumindest der Kern der sog. Von-Neumann-Architektur, die Speicherprogrammierung, eine naturgegebene Sache, die nicht er–, sondern gefunden wird (Bauer 1998, S. 87):

---

53 Möglicherweise stammt der Titel des Papiers gar nicht aus der Feder von Neumanns, sondern wurde von Goldstine eingefügt, der ja das Papier, das er handschriftlich und unfertig von von Neumann erhielt, tippen ließ und anschließend verteilte.

54 Ähnlich stellt (Burks 1987b, S. xv) fest: „Using logical symbolism from the research of Warren McCulloch and Walter Pitts on neural networks in the human brain and such biological vocabulary as ‚organ' and ‚neuron', von Neumann was able to describe the stored-program computer in a way that illuminated the logical functions of the necessary constituent parts without having to consider the particular engineering components that might be employed in any given computer.“

„Noch 1946 bzw. 1947 merkten sie [von Neumann einerseits, Eckert und Mauchly andererseits] nicht, was für ein Goldkorn sie mit dem ‚stored program concept' gefunden (nicht *erfunden*) hatten."

2.2.5 Code

Auch die Konzeption des Codes, wie sie in dem „First Draft" entwickelt wird, spiegelt sich die Hybridobjekt-Sichtweise wider. Der Code erscheint, wie ich zeigen werde, einerseits als technisches Artefakt, welches das Gerät steuert. Gleichzeitig sind auch Einflüsse des mathematischen und des neurophysiologischen Diskurs präsent.

Da mit Hilfe des Codes die Berechnung, die ein Rechengerät durchführen sollte, spezifiziert werden mußte, wird der Code im „First Draft" zunächst vor dem Hintergrund dieses mathematischen Anwendungsfeldes verstanden: Er dient dazu (1.2) „to express the logical and algebraical definition of the problem under consideration, as well as the necessary numerical material." Entsprechend wird in dem Text die mathematische und logische Bedeutung der einzelnen Codewörter beschrieben: Definiert wird (14.1) „the mathematical and logical meaning and the operational significance of its *codewords*." Der Code wird also einerseits aus dem Bereich der Angewandten Mathematik heraus verstanden, er muß die anstehende Berechnung ausdrücken können. Andererseits dient er dazu, das Gerät zu steuern, weshalb auch die „operational significance", seine Bedeutung in bezug auf die ausgelösten Operationen, definiert wird. In dieser Hinsicht wird er als technisches Artefakt betrachtet, wie die folgenden Ausführungen verdeutlichen:

Während die Programmierung des ENIAC durch das manuelle Setzen mechanischer Schalter und das Stecken von Kabelverbindungen erfolgte, sollte ein Programm für den EDVAC in einem elektronischen Speicher abgelegt, d. h. physikalisch durch elektronische Schaltelemente repräsentiert werden. Was zuvor extern mit Hilfe greifbarer technischer Teile repräsentiert wurde, wurde nun zunächst z. B. auf Lochkarten abgebildet und anschließend mit Hilfe geeigneter technischer Vorrichtungen in den Speicher eingelesen. Dazu gibt es nach dem „First Draft" verschiedene Möglichkeiten (1.2):

„These instructions must be given in some form which the device can sense: Punched into a series of punchcards or on teletype tape, magnetically impressed on steel tape or wire, photographically impressed on motion picture film, wired into one or more, fixed or exchangable plugboards – this list being by no means necessarily complete."

Nach dem Erhalt der Instruktionen soll das Gerät diese eigenständig und ohne weitere menschlich-intelligente Hilfe ausführen (1.2): „Once these instructions are given to the device, it must be able to carry the out completely and without any need for further intelligent human intervention." Die Eingabe der Instruktionen kann also über die Imitation verschiedener „Sinne" wie Tasten, magnetisches Abtasten, Sehen recht

beliebig realisiert werden – wichtig ist, daß eine Form gewählt wird, die ein maschinelles Einlesen, ein „Erspüren" durch die Maschine, erlaubt. Die Instruktionen werden hier als materielle Objekte gesehen, die von dem Gerät verarbeitbar sind.[55]

Entsprechend ihrem materiellen Charakter erscheinen in dem „First Draft" die Instruktionen auch Bauteilen der Maschine – den „general control organs", aus denen sich CC, die „Central Control", zusammensetzt – verwandt. Denn man muß die spezifischen Instruktionen, die ein Problem definieren – so von Neumanns hier eingeschobener Hinweis – von den „general control organs" unterscheiden; nur dann wird das Gerät „elastic", soweit möglich ein Allzweckgerät (2. 3):

> „If the device is to be elastic, that is as nearly as possible all purpose, then a distinction must be made between the specific instructions given for and defining a particular problem, and the general control organs which see to it that these instructions – no matter what they are – are carried out. The former must be stored in some way [...] the latter are represented by definite operating parts of the device."

Von Neumann macht also explizit darauf aufmerksam, daß spezifische Instruktionen und Kontrollorgane unterschieden werden müssen. Vor dem Hintergrund, daß vor der Einführung speicherprogrammierbarer Rechenmaschinen Maschinen für spezielle Berechnungen der Regelfall waren, in denen bestimmte Rechenoperationen durch maschinelle Bauteile fest verankert waren, ist der Hinweis auf die Verschiedenheit verständlich. Was nun variabel durch „Instruktionen" bewirkt wird, war zuvor fest in Hardware realisiert. Deutlich wird hier, daß von Neumann die Herkunft der Instruktionen von maschinellen Bauteilen noch deutlich vor Augen stand.

Daß die Instruktionen oder – wie es in dem „First Draft" auch heißt – Befehle als technische, materielle Entitäten gedacht werden, geht auch aus der Art und Weise hervor, wie der Code eingeführt wird (14.1): „It is therefore our immediate task to provide a list of orders which control the device, i.e. to describe the code to be used in the device". Es soll also eine Liste von Befehlen bereitgestellt werden, die das Gerät kontrollieren und die – wie aus der anschließende Ausführung hervorgeht – in dem Gerät benutzt werden. Die Befehle des Codes werden hier nicht außerhalb des Geräts vorgestellt, sondern als Instanzen, denen innerhalb des Geräts dessen Steuerung obliegt. Sie werden damit in ihrer physikalischen Realisierung in dem Gerät gedacht.

Während die Befehle als kontrollierende oder steuernde Instanzen erscheinen, wird CC als die sie ausführende bzw. ihre Ausführung bewirkende Einheit dargestellt (14. 1):

> „The orders which CC receives fall naturally into these four classes: (a) Orders for CC to instruct CA to carry out one of its ten specific operations. (b) Orders for CC to cause the transfer of a standard number from one place to another. (c) Orders for CC to transfer its own

---

55 Gleichzeitig trägt die im „First Draft" gewählte Analogie Gerät-Nervensystem: Ein Instruktionen erhaltender Mensch muß diese ebenfalls zunächst über seine Sinnesorgane erspüren, bevor er sie selbständig ausführen kann.

connection with M to a different point in M, with the purpose of getting its next order from there. (d) Orders controlling the operation of the input and the output of the device."

Alle Befehle sind also an CC gerichtet, die gemäß den erhaltenen Befehlen eine der Operationen von CA, der „Central Arithmetic", anstößt, den Transfer numerischer Daten veranlaßt, den durch die Reihenfolge der im Speicher (M) abgelegten Befehle zunächst gegebenen Befehlsablauf verändert oder den In- und Output kontrolliert.[56] In dem Gerät obliegt also die Steuerung den Befehlen; CC erscheint als nachgeordnete Instanz, die Befehle interpretiert und gegebenenfalls an CA, die in dieser Hinsicht CC nachgeordnet scheint, weitergibt. Dementsprechend bewirkt der Code die logische Kontrolle von CC (2.3): „[...] the code which effects the logical control of CC and hence of the entire device". CC seinerseits führt die logische Kontrolle des Geräts aus und erscheint als Zwischeninstanz zwischen Code und Gerät (2.3): „The logical control of the device, that is the proper sequencing of its operations, can be most efficiently carried out by a central control organ." Unter logischer Kontrolle wird also die korrekte Sequenzierung von Operationen verstanden:[57] Der im Speicher abgelegte Code bestimmt die Reihenfolge der von CC auszuführenden Operationen, CC seinerseits die Reihenfolge der Operationen des Geräts, die im Anschluß an jede von CC aus dem Speicher erhaltene Instruktion ausgeführt werden müssen.

Als die Befehle erhaltende Einheit ist der Aufbau von CC von den möglichen Befehlen, also dem Code, abhängig.

„14.1 Our next aim is to go deeper into the analysis of CC. Such an analysis is, however, dependent upon a precise knowledge of the system of orders used in controlling the device, since the function of CC is to receive these orders, to interpret them and then either to carry them out, or to stimulate properly those organs which will carry them out."

Während also der Aufbau von CC von dem Code abhängt, ergibt sich der Aufbau des Codes, wie das vorletzte längere Zitat (14.1) zeigt, scheinbar selbstverständlich aus dem generellen Aufbau des Geräts: Die Befehle, die CC erhält – also der Code –, unterteilen sich, wie es hier heißt „naturally" (deutsch: natürlich, natürlicherweise) in vier Klassen, die sich aus dem Aufbau des Geräts aus den Komponenten M, CA, CC und Ein-/Ausgabe sowie deren Funktion ergeben. Der Code und der Aufbau der Maschine hängen also wechselseitig zusammen. Während die Konzeption von CC von dem Code abhängt, leitet sich der Aufbau des Codes aus dem generellen Aufbau des Geräts ab.

Einerseits wird also deutlich, daß Code und Gerät in gegenseitiger Dependenz entwickelt werden, was den technischen Charakter des Codes ausdrückt. Das selbstverständliche Ableiten des Codes aus dem Aufbau des Geräts verweist jedoch auch auf

---

56 Eine gut verständliche und detaillierte Darstellung des Codes gibt (Knuth 1970).

57 Die logische Kontrolle stellt also eine Ablaufplanung dar. Das legt die Vermutung nahe, daß die Verwendung des Begriffs „logisch" hier auch in Zusammenhang mit „Logistik" verstanden werden muß.

einen Orientierungsrahmen, vor dessen Hintergrund der technische, an die Maschine gebunden Charakter des Codes quasi unsichtbar wird. Wie ich in den vorangegangenen Abschnitten aufgezeigt habe, ergibt sich die Struktur des Geräts wie selbstverständlich aus seiner Funktion. Auch der Code, der sich seinerseits scheinbar selbstverständlich aus dem Aufbau des Geräts ergibt, leitet sich – dieser Argumentation zufolge – letztlich aus dieser Funktion ab: dem Ausführen von Instruktionen zur Durchführung komplexer Berechnungen. Der Code umfaßt so gesehen die Befehle, die benötigt werden, um komplexe Berechnungen durchführen zu können. Insofern das Gerät als allgemeingültige, dem Ausführen von Instruktion zur Durchführung von Berechnungen unterliegende Struktur konzipiert ist, scheint also der Umstand, daß der Code in einer Maschine realisiert ist, bedeutungslos zu sein.

Dieser Beobachtung entspricht, daß das Gerät in dem „First Draft" einmal als nervensystemähnlicher Organismus, einmal als Maschine erscheint. Ähnlich wie ein in einem Rechenbüro arbeitender Mensch erhält auch das Gerät „Befehle" oder „Instruktionen", die es zunächst in seinem „Gedächtnis" ablegt, anschließend in geeignete Unteroperationen aufsplittet und ausführt. Die ausführende Instanz – ob Mensch oder Maschine – scheint vor dem Hintergrund des neurophysiologischen Diskurses für die Gestalt des Codes tendenziell irrelevant. Auch daß der Code überhaupt als Liste von „Befehlen" gedacht wird – eine Metaphorik, die heute selbstverständlich erscheint –, kann als Ausdruck dieser Analogie Maschine-Nervensystem gesehen werden. Wie ein Mensch oder ein menschlicher Rechner scheint das Gerät dazu in der Lage zu sein, Befehle zu erhalten und auszuführen.

Die für die Konzeption des Geräts konstitutiven Diskurse – der ingenieurwissenschaftlich-technische, der neurophysiologische und der mathematische – spiegeln sich also, wie obige Ausführungen zeigen, auch in der Konzeption des Codes wider. Insoweit das Gerät als technische Maschine gedacht wird, erscheint der Code als technisch-materielles, die Maschine steuerndes Artefakt; Gerät und Code werden in gegenseitiger Abhängigkeit entwickelt. Da er die Durchführung einer Berechnung steuern soll, bezieht er seine Bedeutung nicht nur aus den Operationen, die ein im Speicher abgelegter Befehl bewirkt, sondern auch aus dem Bereich der Angewandten Mathematik: Der Code drückt die „logische" und „algebraische" Definition des zu bearbeitenden Problems aus, den Codewörtern wird entsprechend eine „logische" und „mathematische" Bedeutung zugeschrieben. Das „logisch" muß dabei zwar auch technisch verstanden werden – dem Code obliegt die „logische Kontrolle" des Geräts, d. h. er sorgt, vermittelt über CC, für die korrekte Sequenzierung der Operationen. Versehen mit einer mathematischen und logischen Bedeutung wirken die Instruktionen und Befehle des Codes jedoch auch wie mathematische Anweisungen – zumal das Gerät vor dem Hintergrund des neurophysiologischen Diskurses in Analogie zum menschlichen Nervensystem und damit zum menschlichen, mathematische

Anweisungen erhaltenden Rechner verstanden wird. Soweit das Gerät als Struktur vorgestellt wird, die dem Ausführen von Instruktionen zur Durchführung von Berechnungen unterliegt, erscheint der Code darüber hinaus als Liste von Befehlen, die zur Durchführung von Berechnungen allgemein benötigt werden. Der materiell-technische Charakter tritt in den letzten beiden Hinsichten zugunsten einer mathematischen Sicht auf den Code in den Hintergrund. Auch in der Darstellung des Codes scheint so die Hybridobjekt-Sichtweise auf: Die Befehle des Codes erscheinen zum einen als technische Artefakte, implizit ist der Code jedoch auch konnotiert als Menge von Befehlen zur Durchführung von Berechnungen, die sich prinzipiell genauso an einen rechnenden Menschen richten könnten.

2.3 Erste speicherprogrammierbare Rechenmaschinen

In der Zeit nach der Niederschrift des „First Draft" war von Neumann einerseits in den Rechnerbau involviert und entwickelte in Zusammenarbeit mit anderen einen eigenen Rechner, die IAS-Maschine. In dieser Hinsicht verfolgte er die im „First Draft" deutlich gewordene ingenieurwissenschaftlich-technische Tradition weiter. Auf der anderen Seite setzte er seine Arbeiten zum Vergleich zwischen Rechenmaschine und Nervensystem fort und begann, eine mathematische Theorie zu skizzieren, mit deren Hilfe er beide beschreiben und erfassen können wollte – die Automatentheorie. Er arbeitete also auch die neurophysiologische und eng damit zusammenhängend die mathematische Linie, wie sie sich schon im „First Draft" dokumentieren, weiter aus. In den Abschnitten 2.4 und 3.1 werde ich auf diese beiden Entwicklungen näher eingehen. Zuvor stelle ich kurz den weiteren Werdegang des EDVAC und verwandter Projekte, die von den im „First Draft" geschilderten Ideen beeinflußten wurden, dar, um einen groben Vergleichshorizont zu von Neumanns Texten anzureißen und sie gleichzeitig in einen breiteren Rahmen einzubetten.

Im Mai 1945 kapitulierte Deutschland. Der Krieg ging also in der Zeit, als von Neumann den „First Draft" verfaßte, seinem Ende zu. Das ursprüngliche ENIAC-Team löste sich in der Folgezeit auf. Eckert und Mauchly verließen die „Moore School" und gründeten im Oktober 1946 eine eigene Firma, worauf ich gleich noch eingehe. Goldstine und Burks gingen zu von Neumann an das „Institute of Advanced Studies", wo mit dem Bau einer eigenen Maschine, der IAS-Maschine begonnen wurde (vgl. Abschnitt 2.4). Im Sommer 1946 fand an der „Moore School" noch einmal ein für den weiteren Verlauf der Rechenmaschinenentwicklung sehr einflußreiches Seminar „Theory and Techniques for the Design of Electronic Digital Computers" statt, in dessen Rahmen über die ENIAC-Entwicklung und auch die EDVAC-Konzeption vor 28 Teilnehmern aus Militär, Forschung und Industrie berichtet wurde.

Dieses Seminar wurde von Mitgliedern des Mark I- und des EDSAC-Projektes besucht, in deren Rahmen die ersten speicherprogrammierbaren, elektronischen Rechenmaschinen tatsächlich gebaut wurden. Diese beiden Projekte stelle ich unten kurz dar. Der EDVAC wurde bis 1949 – vermutlich aufgrund der personellen Verschiebungen im ENIAC bzw. EDVAC-Team mit erheblichen zeitlichen Verzögerungen – an der „Moore School" in bezug auf das Design weiterentwickelt und gebaut, um dann an das „Ballistic Research Laboratory" ausgeliefert zu werden. Hier wurden noch bis 1951, dem Jahr, in dem der EDVAC betriebsbereit wurde, Verbesserungen vorgenommen. Der EDVAC blieb im Anschluß bis 1962 in Betrieb (vgl. Randell 1982, S. 378).

Eckert und Mauchly gaben der Firma, die sie 1946 gründeten, vermutlich noch unter dem Eindruck der Kriegs- und Militärerfahrungen zunächst den Namen „Electronic Control Company" und nannten sie schließlich in „Eckert-Mauchly Computer Corporation" um. 1950 wurde diese Firma von der „Remington Rand Division of the Sperry Rand Corporation" erworben. Ihren ersten Rechner, den BINAC, bauten Eckert und Mauchly für die „Northrop Aircraft Company". Diese relativ kleine Maschine, die für die Raketensteuerung eingesetzt werden sollte, arbeitete im August 1949 für einige Stunden korrekt, so daß die „Northrop Aircraft Company" sie akzeptierte und nach Kalifornien verschiffte; danach war sie allerdings nie wieder in vollem Maße funktionstüchtig. Mit ihrer nächsten Maschine, der UNIVAC, waren Eckert und Mauchly erheblich erfolgreicher. Dieser Rechner sollte sowohl für wissenschaftliche wie kommerzielle Aufgaben eingesetzt werden; das erste Exemplar wurde 1951 an das „Bureau of the Census" (statistisches Bundesamt) ausgeliefert, um dort zur Auswertung der Volkszählungsdaten von 1950 verwendet zu werden. Damit wurde ein Rechner zum ersten Mal in großem Umfang nicht für die Durchführung komplizierter wissenschaftlicher Berechnungen, sondern zur Massendatenverarbeitung, bei der einfache Berechnungen auf einer großen Zahl gleich strukturierter Daten durchgeführt werden, eingesetzt. Die UNIVAC dominierte in der Folgezeit den amerikanischen Computermarkt (vgl. Randell 1982, S. 378; Theißing 1995, S. 13f.).

In der US-amerikanischen Öffentlichkeit breit bekannt wurde die neue Maschine spätestens mit dem Einsatz der UNIVAC bei den Präsidentschaftswahlen von 1952. Schon über den ENIAC hatte es einen Wochenschaubericht gegeben, in dem dieser als „electronic brain" bezeichnet worden war – in der Wahlsendung zu der Präsidentschaftswahl wurde die UNIVAC nun durchgehend als eigenständig denkendes Gehirn inszeniert: Die Maschine wurde nicht nur als „electronic brain" bezeichnet; sie wurde zudem von dem Moderator wie eine Person mündlich zu den Wahlergebnissen befragt und gab scheinbar als Antwort auf diese Frage einen Papierstreifen aus. „Als die Berechnungen der UNIVAC dann entgegen allen Vorhersagen einen Erdrutschsieg für

Eisenhower prognostizierten [der auch eintrat], war die Sensation perfekt. Breite Teile der US-amerikanischen Bevölkerung hatten sich selbst von den ‚übermenschlichen Fähigkeiten' dieser Maschine überzeugen können." (Theißing 1995, S. 14) Die Beschreibungsebene, die von Neumann in dem „First Draft" wählte – die Rechenmaschine als aus neuronenartigen Elementen zusammengesetzter Organismus, als eine Art eigenständiges Gehirn – war also auch in der populären Wahrnehmung der neuen Maschinen von zentraler Bedeutung.

Der erste funktionstüchtige speicherprogrammierbare Rechner[58] war wahrscheinlich eine sehr kleine, experimentelle, vorwiegend zum Test eines neuen Speicherprinzips konstruierte Maschine, die 1948 an der Universität Manchester in Großbritannien gebaut und auch erfolgreich getestet wurde. Diese Rechenmaschine diente als Prototyp für die größere und bekanntere Manchester Mark I, die das erste Mal im April 1949 einen erfolgreichen Programmlauf absolvierte, allerdings dabei auf manuell kontrollierten Input und Output angewiesen war. Das Design dieser Rechner schloß, wenn auch nicht ausschließlich, an die Arbeit der „Moore School" an (vgl. Randell 1982, S. 379). Frederic C. Williams, einer der Konstrukteure, hatte das oben erwähnte Seminar an der Moore School im Sommer 1946 besucht. An einem Papier, das den kleinen Prototypen 1948 kurz vorstellt, fällt auf, daß es die Organ-Terminologie des „First Draft" an einer Stelle übernimmt, an der von den „arithmetical organs" (Williams et al. 1948, S. 415) die Rede ist. Ansonsten ist die Sprache rein technisch: Es wird, wie im Englischen im Gegensatz zum US-amerikanischen damals üblich, von „store" statt dem anthropomorphisierenden „memory" gesprochen, und das im „First Draft" sog. „control organ" wird als „control unit" (Williams et al. 1948, S. 415) bezeichnet. Der Initiator des Mark I-Projektes, M. H. A. Newman, war maßgeblich an dem geheimen Bau des Colossus beteiligt gewesen, der im Zweiten Weltkrieg von den Engländern konstruiert wurde, um deutsche Funksprüche zu entschlüsseln. Alan Turing, dem der Erfolg dieses kriegsentscheidenden Vorhabens wesentlich zu verdanken ist, stieß später auch zu dem Manchester-Projekt hinzu. Vielleicht auf Grund von Turings Mitarbeit wird der kleine Prototyp für die MARK I nicht, wie von von Neumann in dem „First Draft", als „elastic" oder „all purpose", sondern wie in Turings Papier von 1936 als „universal" bezeichnet. Dabei wird wie im „First Draft" betont, daß die Instruktionen nicht maschinell festgeschrieben sind, sondern leicht verändert werden können, wobei im Gegensatz zum „First Draft" der Begriff „universal" klarer gefaßt wird (Williams et al. 1948, S. 415):

> „However, apart from its small size, the machine is, in principle, ‚universal' in the sense that it can be used to solve any problem that can be reduced to a programme of elementary instructions; the programme can be changed without any mechanical or electro-mechanical circuit changes."

---

58 Eine knappe Übersicht über die ersten Rechenmaschinen, die man als Von-Neumann-Rechner bezeichnen kann, gibt auch (Bauer 1998, S. 84).

Als erster praktisch einsetzbarer, speicherprogrammierbarer Rechner wird heute allgemein der EDSAC angesehen, dessen Konstruktion Anfang 1947 an der Cambridge Universität, ebenfalls in Großbritannien, begann. Im Mai 1949 wurde das erste Programm erfolgreich ausgeführt. Der Aufbau des EDSAC lehnte sich sehr eng an das EDVAC-Konzept an, so daß die Pläne für den EDVAC als erstes in Großbritannien und nicht den USA umgesetzt wurden. Maurice Wilkes, der leitende Konstrukteur, hatte 1946 die an der „Moore School" gehaltene Vortragsreihe besucht und war, ähnlich wie die meisten Mitglieder des ENIAC-Teams, während des Krieges mit Radartechnologie befaßt gewesen (vgl. Randell 1982, S. 380). Noch weniger als in Manchester machte man sich in Cambridge die biologisierende Metaphorik des „First Draft" zu eigen. In einer Vorstellung des EDSAC von 1950 ist weder von ideellen Elementen, Neuronen oder Organen die Rede, sondern von „machine store, arithmetical unit, input unit, output unit", einem „order interpreter" und einem „input mechanism" (Wilkes et al. 1950, S. 419f.). Zudem beschreibt der Text den technischen Aufbau einer speziellen Maschine, ohne von materiellen Gegebenheiten zu abstrahieren oder übergeordnete Prinzipien anzusprechen. In den ersten Projekten, in denen es gelang, die Speicherprogrammierung technisch zu realisieren, dominierte also eine ingenieurwissenschaftlich-technische, auf den Bau einer speziellen Maschine ausgerichtete Kultur.

2.4 Die IAS-Maschine

Ende 1945 entschied von Neumann, an seinem Institut, dem „Institute of Advanced Studies" in Princeton, einen eigenen Computer zu bauen, der unter der Bezeichnung IAS-Maschine oder auch Von-Neumann-Maschine bekannt geworden ist. Obwohl sich dieses Anliegen in den im Rahmen dieser Rechenmaschinenentwicklung von von Neumann und seinen Mitarbeitern verfaßten Schriften nicht so deutlich dokumentiert wie in dem „First Draft", ging es auch hier darum, das Nervensystem und Gehirn zumindest in Teilen nachzubilden. So schrieb Frank Aydellote, der damalige Direktor des „Institute of Advanced Studies" (nach McCorduck 1979, S. 63): „Curiously enough, the plan of such a machine is partly based on what we know about the operation of the central nervous system in the human body." Ähnlich erinnerte sich von Neumanns Frau, Klara von Neumann (1958, S. 10): „Beim Entwurf der Maschine versuchten John und seine Mitarbeiter, einige der bekannten Vorgänge im lebenden Gehirn zu imitieren" (Klara von Neumann 1958, S. 10).

Von Neumann engagierte sich also auch nach 1945 im Rechenmaschinenbau und machte sich, so jedenfalls Burks, auch als Erfinder einen Namen:

„As well as being an outstanding physicist, mathematician, and logician, von Neumann was an inventor. In late summer 1944, he conceived of using an electron-beam-oscilloscope for computer storage" (Burks 1987a, S. 12f.).

Basierend auf dieser neuen Speichertechnologie sollte die IAS-Maschine Daten und Befehle nicht mehr streng seriell, bit für bit, sondern parallel verarbeiten. Bei dem Versuch, ein Datensortierproblem zu programmieren,[59] hatte von Neumann nämlich festgestellt, daß das Programm bei serieller Speicherung zu lange darauf warten mußte, daß die benötigten Daten aus dem Speicher in die verarbeitenden Einheiten und zurück transferiert wurden. Bereits im „First Draft" hatte von Neumann die Möglichkeit, einen „iconoscope memory" zu verwenden, andiskutiert und insbesondere auch den Vorteil einer solchen Speichertechnologie hervorgehoben, jederzeit schnell zu jedem beliebigen Teil des Speichers springen zu können. Die IAS-Maschine nun sollte die Möglichkeit einer solchen Technologie, Daten und Befehle parallel zu verarbeiten, ausschöpfen (Burks 1987a, S. 13): „The bits of a word were to be stored in parallel, transmitted in parallel, and processed in parallel." Die technische Umsetzung einer parallelen Speichertechnologie wurde jedoch aus dem Institut ausgelagert. Man beauftragte zunächst die „Princeton Laboratories of the Radio Corporation of America" mit der Herstellung von „electrostatic tubes called selectrons". Der Versuch schlug allerdings fehl, woraufhin auf eine andere Erfindung – „Standard cathode-ray tubes for doing binary information" (Burks 1987a, S. 15), wie sie F. C. Williams, Mitarbeiter in dem erwähnten Manchester-Projekt, entwickelt hatte – zurückgegriffen wurde.

Wie schon diese Episode vermuten läßt, lag der Hauptfokus von Neumanns und der IAS-Gruppe weniger im Bereich der technischen Konstruktion. Zwar gab es eine Ingenieursgruppe, die nacheinander von Julian Bigelow, der im Krieg mit Wiener zusammengearbeitet hatte,[60] James Pomere und Willis Ware geleitet wurde (vgl. Burks 1987a, S. 15). Die Gruppe war also personell relativ unbeständig, was vielleicht auch darauf zurückzuführen ist, daß sie die Ideen einer anderen Gruppe, die vorwiegend aus Mathematikern zusammengesetzt war – von Neumann sowie Burks und Goldstine –, lediglich „in ‚Hardware' übersetzen" sollte (Goldstine 1979, S. 61).[61] Die relativ geringe Technikbezogenheit der IAS-Gruppe mag dazu beigetragen haben, daß die IAS-Maschine erst 1952 fertiggestellt wurde. Zu dieser Zeit war der EDSAC schon drei Jahre lang funktionstüchtig, und Eckert und Mauchly hatten die UNIVAC bereits

---

59 Dieses Programm wird ausführlich in (Knuth 1970) besprochen.

60 Vgl. dazu auch Abschnitt 3.1.1.

61 Eine dritte Gruppe war mit numerischer Analysis befaßt und entwickelte solche numerischen Techniken (weiter), wie sie für die Programmierung der neuen, digitalen Rechenmaschinen benötigt wurden. Eine vierte Gruppe war, nach (Goldstine 1979, S. 63) sehr erfolgreich, mit mathematischer Meteorologie, einem Steckenpferd von Neumanns, beschäftigt. Mathematische Arbeiten standen also am IAS stark im Vordergrund.

erfolgreich auf dem US-amerikanischen Markt plaziert. Die IAS-Machine wurde nach ihrer Fertigstellung beim Bau der Wasserstoffbombe, an dem auch von Neumann beteiligt war, eingesetzt.

Die IAS-Maschine lief also erst recht spät, so daß es nicht in erster Linie das technische Artefakt selbst sein kann, das für die weitere Entwicklung der Rechenmaschinen große Bedeutung hatte. Sehr großen Einfluß hatten jedoch die Berichte[62], die Burks, Goldstine und von Neumann ab dem Frühjahr 1946 verfaßten (Randell 1982, S. 379):

> „Although the IAS computer was not finished until 1952, the series of reports that were issued by the project, starting in 1946, were widely circulated and served many people as textbooks on logical design and programming. As a result many similiar projects for parallel binary machines, or von Neumann machines as they came to be known, were started up. One such project was within IBM, and resulted in the IBM 701, forerunner of a whole series of machines which within a few years became the dominant large-scale scientific computers."

Lange bevor die IAS-Maschine fertiggestellt war, entfalteten von Neumann und seine Mitarbeiter in der Entwurfsgruppe also eine rege Publikationstätigkeit (Heintz 1993, S. 226):

> „Keine andere Gruppe entwickelte eine vergleichbare publizistische Aktivität, und keine andere Gruppe hatte eine vergleichbare Stellung inne, was wissenschaftliches Ansehen und Kommunikationsressourcen betraf. Diese diskursiven Kompetenzen – und nicht technische Faktoren allein – haben den Verlauf des Selektionsprozesses entscheidend mitbestimmt. ‚His scientific standing', so William Aspray über John von Neumann, ‚was an important factor in winning over the scientific and funding communities, and it was in stark contrast to that of most of the other pioneers of modern computing.' (Aspray 1990, S. 4f.)"

Neben Publikationen spielten auch persönliche Kontakte, über die von Neumann auf Grund seiner exponierten Stellung im Wissenschaftsbetrieb in großer Zahl verfügte, für die Verbreitung der Ideen der IAS-Gruppe eine große Rolle (Heintz 1993, S. 221f.):

> „Neben Forschungsberichten waren in diesen ersten Jahren persönliche Kontakte das Hauptmedium der Wissensdiffusion, und auch in dieser Hinsicht hatte das IAS-Team eine herausragende Stellung inne. Der IAS-Computer stand schon bald im Mittelpunkt der Tagungen, die man nach Beendigung des Krieges zu organisieren begann [...]. Wer sich mit Computern befaßte, hat zu dieser Zeit das Institute of Advanced Studies besucht und sich von John von Neumann und seinen Mitarbeitern die Konstruktionsdetails des IAS-Computers persönlich erklären lassen."

Es waren also mehr die Ideen, die im Rahmen des Vorhabens, die IAS-Maschine zu bauen, in Berichtsform ausgearbeitet wurden, die großen Einfluß auf die weitere Rechenmaschinenentwicklung nahmen – und das nicht zuletzt, weil die Stellung von Neumanns und seiner Gruppe im Wissenschaftsbetrieb die weite Verbreitung dieser Ideen garantierte.

---

62 Dabei handelt es sich um (Burks, Goldstine, von Neumann 1946), (Goldstine, von Neumann 1947, 1948a, 1948b).

2.4.1 Text: „Preliminary Discussion of the Logical Design of an Electronic Computing Instrument“

Die IAS-Berichte, und besonders der erste, hatten nach Einschätzung von Burks noch größere Bedeutung als der „First Draft of a Report on the EDVAC“:

> „In the spring of 1946, von Neumann, Goldstine and I worked out the logical design of the processor and memory of the IAS computer. Our report, Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, was issued in June 1946, and a revised edition was issued in September 1947. The report was widely distributed and was even more influential than von Neumann's EDVAC report.“ (Burks 1987a, S. 13)

Der „Preliminary Discussion“ ist deutlich anzumerken, daß sie in Folge des „First Draft“ entstand. Schon die äußere Form der beiden Papiere, die Aufteilung in relativ kleine, durchnumerierte Abschnitte, macht darauf aufmerksam. Auch der generelle Aufbau des beschriebenen Geräts – „memory-“, „control-“, „arithmetic-“ sowie „input/output-organ“, duale Daten- und Befehlsdarstellung – ist im wesentlichen derselbe. In beiden Papieren wird großer Wert auf den automatischen Charakter der Maschine gelegt und ebenfalls in beiden die Ein- und Ausgabe nur sehr rudimentär behandelt. Während der „First Draft“ jedoch, den einleitenden Worten zufolge, die Struktur eines digitalen Hochgeschwindigkeitsrechensystems behandelt, geht es in der „Preliminary Discussion“ um „some aspects of the overall logical considerations arising in connection with electronic computing machines“, so das Vorwort zur ersten Ausgabe. Die verschiedenen Formulierungen verweisen auf einen wesentlichen Unterschied zwischen den beiden Papieren: Während es im „First Draft“ um die Struktur eines Systems geht, fokussiert die „Preliminary Discussion“ stärker auf den Maschinenaspekt. Seinen Ausdruck findet dieser andere Schwerpunkt darin, daß die „Preliminary Discussion“ sich eine geringere Allgemeinheitsebene zu eigen macht. Zwar werden häufig mehrere mögliche technische Umsetzungen für eine bestimmte Einheit andiskutiert (z. B. in 5.11 für den Akkumulator oder in 4.1 für den Speicher) und die letztendliche technische Gestalt nicht vollständig festgelegt, jedoch werden keine ideellen Elemente eingeführt, die für digitale Elemente allgemein und auch für Neuronen stehen können. Entsprechend fällt der im „First Draft“ durchgeführte Vergleich zwischen dem beschriebenen Gerät bzw. seinen Einheiten mit dem Nervensystem in der „Preliminary Discussion“ weg, auch wenn die Organterminologie beibehalten wird.[63] Während von Neumann den „First Draft“ als Berater eines Projektes schrieb, in dem der Bau der anvisierten Maschine, des EDVAC, noch in Planung war, waren er, Burks und Goldstine nun mit dem Bau der IAS-Maschine

---

63 Auffällig ist in diesem Zusammenhang auch, daß in der „Preliminary Discussion“ zwar nach wie vor das antropomorphisierende „memory“ (Gedächtnis) benutzt wird, jedoch – anders als im „First Draft“ – auch häufig auf Begriffe wie „store“ oder „storage“ (Speicher(ung)) zurückgegriffen wird.

direkt befaßt. Die stärkere Fokussierung auf den Maschinenaspekt ist, so meine Vermutung, ein Ausdruck dieses anderen Enstehungszusammenhangs der „Preliminary Discussion".

Daß es in der „Preliminary Discussion" stärker um die konkrete maschinelle Umsetzung geht, paßt dazu, daß der Gegenstand des Papiers nicht, wie zu Beginn des „First Draft", erst definiert wird.[64] Statt dessen wird auf Institutionen verwiesen, die bestimmte Bauteile herstellen (die Princeton RCA Labs, 4.1), sowie auf Erfahrungen beim Bau mit einer anderen Maschine, dem EDVAC (vgl. 4.1, 4.2). Die „Preliminary Discussion" erscheint damit eher als Diskussion einer Maschine, die eine bestimmte Geschichte hat.

Deutlicher als im „First Draft" dokumentiert sich in der „Preliminary Discussion", daß die Struktur der Maschine von den verwendeten technischen Bauteilen abhängt. So schreiben die Autoren hier (3.7):

> „Inasmuch as our experience has shown that the moment one chooses a given component as the elementary memory unit, one has also more or less determined upon much of the balance of the machine, we start by a consideration of the memory organ."

Es wird damit die Abhängigkeit der gesamten Planung von der Entscheidung für eine bestimmte Speichertechnologie betont – sicherlich ein Ausdruck der Erfahrung, daß die Umstellung von der seriellen Speichertechnologie des EDVAC, den „mercury delay lines", zu der parallelen Speichertechnologie, so wie sie für die IAS-Maschine vorgesehen war, nicht unerheblich auch das Design der anderen Organe beeinflußte. Denn auch diese sollten nun die eingehenden Daten und Befehle soweit möglich parallel verarbeiten. Neben der Feststellung der Abhängigkeit des Aufbaus von den materiellen Gegebenheiten findet sich eine zentrale Argumentationsstruktur des „First Draft", nach der sich der Aufbau des Geräts aus seiner in der Definition festgelegten Funktion ergibt, auch in der „Preliminary Discussion" wieder, allerdings in abgeschwächter Form. Der erste Satz des ersten Abschnitts lautet:

> „1.1 Inasmuch as the completed device will be a general-pupose computing machine it should contain certain main organs relating to arithmetic, memory-storage, control and connection with the human operator."

Auch hier wird aus der Funktionalität des Geräts, eine universelle Rechenmaschine zu sein, der generelle Aufbau abgeleitet, so daß dieser seinen die jeweilige stoffliche Ausgestaltung transzendierenden Charakter behält, den er auch im „First Draft" hatte. Das „should" verleiht dem Aufbau jedoch normativen und empfehlenden Charakter. Das verweist darauf, daß für die Autoren auch andere Möglichkeiten des Aufbaus denkbar sind.

Daß in der „Preliminary Discussion" der Maschinenaspekt stärker im Vordergrund steht als in dem „First Draft", spiegelt sich auch in der Darstellung des Codes wider.

---

64 Darauf bin ich in Abschnitt 2.2.3 genauer eingegangen.

Dieser erscheint als in Abhängigkeit von den maschinellen Gegebenheiten und seinem Anwendungsgebiet zu gestaltendes technisches Artefakt. Explizit wird hier auf die Herkunft der Instruktionen von maschinellen Bauteilen hingewiesen (1.2):

> „It is evident that the machine must be capable of storing in some manner [...] the instructions which govern the actual routine to be performed on the numerical data. In a special-purpose machine these instructions are an integral part of the device and constitute a part of its design structure. For an all-purpose machine it must be possible to instruct the device to carry out any computation that can be formulated in numerical terms."

Die Instruktionen werden also als gespeicherte, den Bauteilen einer Maschine für spezielle Berechnungen verwandte, technisch realisierte Entitäten gesehen. Sie scheinen es zu sein, denen die Kontrolle in der Maschine obliegt. Dem Zitat zufolge regeln bzw. steuern sie die aktuelle Routine, die auf den im Speicher abgelegten numerischen Daten ausgeführt werden soll. An anderen Stellen

- ist es ähnlich ein Befehl „which will, depending on the sign of a given number, cause the proper one of two routines to be executed" (3.5);
- sind es Befehle „which can instruct the machine to read or write on the medium" (4.5).

Deutlich wird jedoch auch, daß die steuernde Kraft nicht allein von den Befehlen ausgeht, sondern erst im Zusammenspiel von Code und „control" – wie die zentrale Steuereinheit in der „Preliminary Discussion" genannt wird – entsteht (6.1):

> „It has already been stated that the computer will contain an organ, called the control, which can automatically execute the orders stored in the Selectrons. [...] Let us consider the routine that the control performs in directing a computation. The control must know the location in the Selectron memory of the pair of orders to be executed. It must direct the Selectrons to transmit this pair of orders to the Selectron register and then to itself."[65]

Einerseits führt hier die „control" die gespeicherten Befehle aus, andererseits weist sie den Speicher an, ihr Befehle zu schicken. Die Steuerung der Maschine ist also ohne eine initiierende Operation der „control" nicht möglich.

Entsprechend diesem Zusammenspiel von Code und „control" bei der Steuerung werden der Aufbau der Maschine und des Codes in gegenseitiger Abhängigkeit entwickelt. Auf der einen Seite ergibt sich der Aufbau des Codes aus dem generellen Aufbau der Maschine. Wie in der „Preliminary Discussion" der Aufbau der Maschine sehr viel vorsichtiger aus der Funktion abgeleitet wird als im „First Draft", wird der Aufbau des Codes vorsichtiger aus dem generellen Aufbau der Maschine abgeleitet:

> „3.2 There must certainly be instructions for performing the fundamental arithmetic operation. [...] 3.3 It must be possible to transfer data from the memory to the arithmetic organ and back again."

---

65 Ähnlich heißt es in (6.4): „It should be noted that all these operations are directed by the control itself – not only the operations specified in the control words sent to FR [Function Table Register], but also the automatic operations required to get the correct orders there."

Zudem wird betont, daß nur ein spezieller Typ Code besprochen wird – der für die eigene Maschine vorgesehene (vgl. 3.1). Dadurch wird die Gültigkeit der folgenden Überlegungen explizit auf diesen Typ Code eingeschränkt.

Während also einerseits der Aufbau des Codes von dem der Maschine abhängt, erscheint umgekehrt auch der Aufbau der Maschine vom Code abhängig; ein bestimmter Code, so heißt es in folgendem Zitat, erfordert eine bestimmte Ausrüstung der Maschine (3.1):

> „The really decisive considerations from the present point of view, in selecting a code, are more of a pratical nature: simplicity of the equipment demanded by the code, and the clarity of its application to the actually important problems together with the speed of its handling of those problems."

Neben der Maschine wird hier auch die Handhabbarkeit des Codes als wichtiger Faktor für die Gestaltung des Codes benannt. Auch der Problembereich, für den der Code verwendet werden soll, wird damit als Faktor in der Codegestaltung explizit miteinbezogen. Der Code erscheint also als technisches, in Zusammenhang mit der „control" die Maschine steuerndes Artefakt, das in wechselseitiger Abhängigkeit mit der Maschine und mit Bezug auf den Anwendungsbereich gestaltet wird.

Der Zusammenhang zwischen Code und Maschine wird in der „Preliminary Discussion" vor allem als gegenseitige technische Abhängigkeit gedacht. Deutlich wird jedoch auch der Einfluß mathematischer Diskurse auf die Darstellung des Codes (3.1):

> „It is easy to see by formal-logical methods that there exist codes that are *in abstracto* adequate to control and cause the execution of any sequence of operations which are individually available in the machine and which are, in their entirety, conceivable by the problem planner."

Der Code wird in dem Zitat als Gegenstand formal-logischer Überlegungen gesehen: Die Existenz eines adäquaten Codes für eine Maschine läßt sich, so die Autoren, formal-logisch beweisen. Der Zusammenhang zwischen Code und Maschine wird also nicht nur rein technisch gedacht, sondern auch als Gegenstand mathematisch-logischer Methoden.

Daß mathematische und technische Betrachtungsweise in der „Preliminary Discussion" eng miteinander verknüpft sind, macht auch die Darstellung der Bedeutung des Codes deutlich. Ähnlich wie in dem „First Draft" von der „mathematical and logical meaning" der einzelnen Codewörter die Rede ist, wird in der „Preliminary Discussion" die Bedeutung der Codesymbole als mathematisch bezeichnet. Die Codesymbole werden also auch aus dem Anwendungsgebiet des Codes, der Steuerung und Ausführung von Berechnungen, heraus verstanden (6.6):

> „In order to have the precise mathematical meaning of the symbols which are introduced in what follows clearly in mind, the reader should consult the table at the end of the report for each new symbol, in addition to the explanations given in the text."

In der Beschreibung der Bedeutung der einzelnen Symbole des Codes in der in dem Zitat angesprochenen Tabelle wird eine aus dem mathematischen Bereich entlehnte Terminologie mit einer technischen, auf den Aufbau der Maschine bezogenen, Begrifflichkeit gemischt – so z. B. in den folgenden Zeilen, die der Tabelle entnommen sind:

S(x) -> Ac+	Clear accumulator and add number located at position x in the Selectrons into it
A ÷ S(x) -> R	Clear register and divide the number in the accumulator by the number located in position x in the Selectrons, leaving the remainder in the accumulator and placing the quotient in the register

Zu den zu verarbeitenden Zahlen muß also die technische Komponente angegeben werden, in der sie sich in der Maschine befinden bzw. in die sie transferiert werden sollen. Die Zahlen erscheinen hier, wie oben die Instruktionen, als technisch verarbeitbare Artefakte. Die Operationen der Komponenten des „arithmetic organ" werden durch ihre mathematische Entsprechung, hier das Addieren bzw. das Dividieren, gefaßt. Die „mathematische Bedeutung" des Codes wird also durch Rückbezug auf den technischen Aufbau der Maschine beschrieben. Die mathematische und die technische Betrachtung des Codes sind eng miteinander verwoben.

Der neurophysiologische Diskurs, der im „First Draft" eine zentrale Rolle spielt, dokumentiert sich in der „Preliminary Discussion" neben der verwendeten Organbegrifflichkeit für die Beschreibung der Rechenmaschine in der Metaphorik, die zur Beschreibung des Codes und seines Zusammenwirkens mit der „control" verwendet wird. Das Kodieren wird hier im Vorwort – nur dieses eine Mal – als Aufbereiten mathematischer Probleme in der „Sprache" der Maschine eingeführt:

> „An attempt is made to give [...] in the second half a study of how actual mathematical problems can be coded, i. e. prepared in the language the machine can understand."

Die „control" erscheint im Zusammenhang mit der Einführung der Sprachmetaphorik als das Organ, das die Befehle „versteht" (1.2): „There must, moreover, be a unit which can understand these instructions and order their execution." Die Befehls- und Instruktionsmetaphorik des „First Draft" wird hier ausgeweitet: Der Maschine wird die Fähigkeit, eine Sprache zu verstehen, zugesprochen, wie es für ein Gerät, das als eine Art Organismus mit Ähnlichkeit zum menschlichen Nervensystem und Gehirn gesehen wird, naheliegt. Die Ein-/Ausgabeorgane werden entsprechend als Vorrichtungen beschrieben, mit deren Hilfe Mensch und Maschine miteinander kommunizieren (1.6): „Lastly there must be devices, the input and output organ, whereby the human operator and the machine can communicate with each other." In dem „First Draft" bestand die Aufgabe der Ein-/Ausgabeorgane darin, das „Spüren" (engl. sense) der Eingabe und besonders der eingegebenen Instruktionen durch das Gerät bzw. der Ausgabe durch

menschliche Organe zu ermöglichen; die Metaphorik lehnte sich also an den körperlich-sinnlichen Bereich an. Maschine und Gerät schienen in gleicher Weise zu fühlen bzw. zu tasten. In der „Preliminary Discussion“ wird diese Begrifflichkeit durch eine Metaphorik ersetzt, die aus dem Bereich der sprachlichen Verständigung entnommen ist. Mensch und Maschine erscheinen als mehr oder weniger gleichrangige Kommunikationspartner.

Die „Preliminary Discussion“ erscheint im Vergleich zum „First Draft“ stärker als Besprechung materieller, hergestellter und geschichtlicher Maschinen und des zugehörigen Codes. Im Vergleich mit dem „First Draft“ erscheint die „Preliminary Discussion“ zwar relativ maschinennah; im Vergleich mit anderen Texten zum Rechenmaschinenbau der Zeit wird jedoch auf einer recht allgemeinen Ebene diskutiert. Wie schon die Überschrift ankündigt, soll das „logische Design“ eines elektronischen Recheninstruments betrachtet werden – oder, wie es etwas später heißt: „die Gesamtheit betreffende logische Überlegungen im Zusammenhang mit elektronischen Rechenmaschinen“ (Vorwort). Wichtig war die Ausarbeitung allgemeinerer Prinzipien, die auch für andere Rechnerbauprojekte interessant sein sollten. Auch wenn technische Überlegungen im Vordergrund stehen, sind sie eng mit Betrachtungsweisen verknüpft, die als mathematisch oder logisch bezeichnet werden. Der dem „First Draft“ unterliegende neurophysiologische Diskurs drückt sich in der „Preliminary Discussion“ in der verwendeten Metaphorik aus, die nun auch dem Bereich der sprachlichen Verständigung entstammt. Der Code erscheint so einerseits als technisches, die Maschine steuerndes Artefakt; andererseits wird er nicht mehr nur als Liste von Befehlen mit mathematischer Bedeutung, sondern allgemeiner auch als eine Art Sprache gesehen, die die Maschine bzw. C versteht.

2.4.2 Text: „Planning and Coding Problems for an Electronic Computing Instrument“

Ungefähr neun Monate nach der „Preliminary Discussion“ stellten Goldstine und von Neumann einen Bericht „of our methods of coding and of the philosophy which governs it“ (Goldstine, von Neumann 1947, Preface) vor: den ersten Teil der „Planning and Coding Problems“, dem 1948 zwei weitere Teile folgten. Burks, Mitkonstrukteur des ENIAC, war mittlerweile an die Universität Michigan berufen worden. Er verfaßte diese Papiere nicht mehr mit. Auch die „Planning and Coding Problems“ wurden weit verbreitet und waren für das entstehende Wissensgebiet der Programmierung elektronischer Rechenanlagen fundamental – zumal sie bis 1951 die fast einzigen im

englischen Sprachraum erhaltbaren Dokumente zum Thema Programmiermethodik waren (Knuth, Trabb Pardo 1980, S. 208):[66]

> „the treatise of Goldstine and von Neumann was [...] distributed in quantity to the vast majority of people involved with computers at that time. This fact, coupled with the high quality of presentation and von Neumann's prestige, meant that their report had an enormous impact, forming the foundation for computer programming techniques all over the world."

Während es im „First Draft" und der „Preliminary Discussion" um den Aufbau der Rechenmaschine und des Codes ging, war es nun das Kodieren – also die Arbeit mit dem zuvor erstellten Code –, das im Zentrum der Aufmerksamkeit stand (Goldstine, von Neumann 1947, 7.1, S. 152): „In this, the second part of the report, we intend to show how the orders may be used in the actual programming of numerical problems."

Das Kodieren beschreiben die Autoren als einen Prozeß, der regelhaft und unabhängig von einer bestimmten Maschine vier Stufen durchläuft (vgl. 7.9): Dem Kodieren voran geht demnach eine erste, rein mathematische Phase, die mit der Maschine nichts zu tun hat und auch für von Hand zu erledigende Rechnungen nötig ist. Das eigentliche Kodieren beginnt anschließend mit dem Zeichnen von Flußdiagrammen, der sogenannten dynamischen oder makroskopischen Stufe. In der dritten, der statischen oder mikroskopischen Stufe, werden diese in Code übersetzt und in der letzten Phase die Speicheradressen zugeordnet. Die letzten drei Phasen, so wurde von den Autoren wiederholt versichert, seien einfach und mit ein wenig Übung leicht zu bewältigen.

Ich werde im folgenden besonders die ersten beiden Phasen näher betrachten. Dabei zeigt sich, daß das Kodieren einerseits als von der Maschine nicht zu trennende und in diesem Sinne technische Tätigkeit konzipiert ist. Das entspricht der Sicht auf den Code als technisches, die Maschine in Zusammenarbeit mit der „control" steuerndes Artefakt, die ich im letzten Abschnitt herausgearbeitet habe. Ähnlich wie die Rechenmaschine und der Code wird das Kodieren andererseits auch aus einer mathematischen Perspektive betrachtet, aus der die Maschine in ihrer Bedeutung für das Kodieren unsichtbar wird.

In der den „Planning and Coding Problems" zufolge ersten Phase, der mathematischen Vorbereitung der eigentlichen Kodierung, soll das zu bearbeitende Problem zunächst „mathematisch" oder „mathematisch-physikalisch" verstanden werden, wobei auch zu entscheiden ist, auf Grundlage welcher Annahmen und Idealisierungen Gleichungen und Bedingungen aufgestellt werden. Sie wird von einem „planner" durchgeführt (vgl. 7.9, S. 171). Diese Arbeit wird als rein mathematisch gesehen und hat weder Bezug zum Rechnen noch zur Maschine (7.9, S. 170): „It should be noted that the first step

---

66 Vgl. dazu auch (Burks 1987e, S. 146).

has nothing to do with computing or with machines: it is equally necessary in mathematics or applied mathematics."

Weil eine solche mathematische Problembeschreibung, so die Autoren, in der Regel Gleichungen und Bedingungen enthält, die „analytischer" und „impliziter" Natur sind, müssen sie anschließend in „arithmetische" und „explizite" Vorgänge aufgebrochen werden.[67] Auf Grundlage numerischer Methoden wird dabei ein konstruktives Rechenverfahren erstellt. Abschließend werden, wie bei numerischen Berechnungen üblich, Fehler abgeschätzt, die durch die gewählten Näherungsverfahren oder als Rundungsfehler in der Maschine auftreten können. Auch dieser Schritt der Erstellung eines konstruktiven Rechenverfahrens, so betonen die Autoren, verdankt sich nicht dem Umstand, daß eine Maschine programmiert werden soll, sondern ergibt sich aus der Notwendigkeit des zahlenmäßigen Berechnens (7.9, S. 171):

> „Furthermore, the second step has, at least, nothing to do with mechanization: it would equally be necessary if the problems were to be computed ‚by hand'. [...] it is necessary because of the computational character of the problem, rather than because of the use of a machine."

Die Angabe eines solchen, auf numerischen Methoden basierenden Rechenverfahrens war auch dann nötig, wenn die Berechnungen in einem Rechenbüro durchgeführt wurden. Die Phase der mathematischen Problemvorbereitung stellt sich so als unabhängig von der Rechenmaschine dar.

In der zweiten – der dynamischem oder makroskopischen – Phase wird im Gegensatz zu der ersten Phase der Umstand, daß die anstehende Berechnung von einer Maschine ausgeführt werden soll, zu einem entscheidenden Faktor. Mit ihr beginnt den „Planning and Coding Problems" zufolge das eigentliche Kodieren, das von einem „coder" durchgeführt wird (7.9, S. 171):

> „After these preliminaries [the mathematical stage of preparations] are completed, the coding proper can begin. [...] Coding begins with the drawing of the flow diagrams. This is the dynamic or macroscopic stage of coding."

Die dynamische Phase besteht in dem Zeichnen von Flußdiagrammen, die ein Zwischenglied zwischen der mathematischen Aufbereitung des Problems und dem letztendlichen Code bilden.[68] Ihre Notwendigkeit oder zumindest Vorteilhaftigkeit begründen die Autoren mit der „Natur des Kodierens": Kodieren ist ihnen zufolge nicht – wie es der Name nahelegen könnte – die Übersetzung eines Textes aus der „Sprache der Mathematik" in die „Sprache des Codes". Diese Feststellung ist für ihre

---

67 Zur Erläuterung heißt es (7.9, S. 170): „These are usually step-by-step processes or successive approximation processes, or processes with both of theses characteristics – and they are almost always characterized by multiple inductions."

68 Das Erstellen von Flußdiagrammen wird damit explizit als Teil des Kodierens und nicht etwa als dessen Vorbereitung gesehen, wie das heute üblich ist.

weiteren Ausführungen und das Verständnis der Flußdiagramme zentral. Zudem wird deutlich, daß und worin die Autoren prinzipielle Unterschiede zwischen dem mathematischen Text, wie er in der ersten Phase erstellt wird, und einer zugehörigen Codesequenz[69] sehen und was damit die „Natur des Kodierens" ausmacht. Ich werde deshalb näher auf ihre diesbezüglichen Ausführungen eingehen, bevor ich auf die Flußdiagramme zu sprechen komme. Die Regeln und Notationsweisen, wie sie Goldstine und von Neumann für ihre Flußdiagramme einführen, sind in (Knuth, Trabb Pardo 1980) für heutige Leser gut verständlich dargestellt.[70] Ich werde mich hier auf prinzipiellere Überlegungen in bezug auf die Tätigkeit des Kodierens und die Rolle von Flußdiagrammen beschränken, da es mir um den Orientierungsrahmen geht, vor dessen Hintergrund diese Arbeit gesehen wurde.

Während der Phase der mathematischen Problemvorbereitung soll, wie oben dargestellt, ein konstruktives Rechenverfahren erstellt werden. Dieses muß in den folgenden Phasen in eine Codesequenz überführt werden, die die Maschine veranlaßt, die gewünschten, das Problem lösenden Operationen durchzuführen. Diese Codesequenz wird, wie das bereits in der „Preliminary Discussion" deutlich wurde, auch in den „Planning and Coding Problems" in ihrer Eigenschaft als technische Steuerungsinstanz der Maschine gesehen (7.2, S. 152):

> „The actual code for a problem is that sequence of coded symbols [...] that has to be placed into the Selectron memory in order to cause the machine to perform the desired and planned sequence of operations, which amount to solving the problem in question."

Würden nun diese im Speicher abgelegten Instruktionen linear abgearbeitet, könne man, so die Autoren, das Kodieren als Übersetzen zwischen zwei Sprachen – der der Mathematik in die des Codes – verstehen (7.1, S. 152):

> „Coding a problem for the machine would merely be what the name indicates: Translating a meaningful text (the instructions that govern solving the problem under consideration) from one language (that of mathematics, in which the planner will have conceived the problem or rather the numerical procedure by which he has decided to solve the problem) into another language (that one of our code)."

Da jedoch bei der Abarbeitung einer im Speicher abgelegten Codesequenz die Kontrolle C (die Organterminologie wird in diesem Papier nicht verwendet) auf Grund von Sprungbefehlen die Reihenfolge der im Speicher abgelegten Instruktionen nicht einhalte, darüber hinaus im Falle bedingter Sprungbefehle diese Veränderung der Reihenfolge von erst während der Rechnung erzielten Zwischenresultaten abhänge und

---

69 Der Begriff Programm wird in allen in diesem Kapitel von mir besprochenen Texten nicht verwendet. Um eine möglichst unverfälschten Widergabe zu gewährleisten, werde auch ich ihn nicht benutzen.

70 Eulenhöfer (1998b, S. 112f.) stellt die berechtigte Vermutung an, daß die Art der graphischen Darstellung von Flußdiagrammen an die Neuronenkreismodelle angelehnt ist, wie sie damals in der Neurophysiologie diskutiert wurden und wie sie von Neumann (1945) in ähnlicher Form zur Beschreibung des Rechengeräts verwendet hatte (vgl. dazu Schmidt-Brücken 1998).

zudem der Code selbst während des Maschinenlaufs verändert werden könne, sei im allgemeinen der Lauf von C durch den Code nicht vorhersehbar (7.1, S. 153):

> „C will, in general, not scan the coded sequence of instructions linearly. It may jump occasionally forward or backward, omitting [...] some parts of the sequence, and going repeatedly through others. It may modify some parts of the sequence while obeying the instructions in another part of the sequence. [...] All these displacements and modifications may be conditional upon the nature of intermediate results obtained by the machine itself in the course of this procedure. Hence, it will not be possible to foresee in advance and completely the actual course of C".

Der Zusammenhang zwischen den kodierten Instruktionen und der mathematischen Problembeschreibung der ersten Phase kann aus diesem Grund – daß die im Speicher abgelegte Codesequenz nicht linear bearbeitet wird – nicht als ein statischer, sondern muß als ein dynamischer verstanden werden (7.1, S. 153):

> „Thus the relation of the coded instruction sequence to the mathematically conceived procedure of (numerical) solution is not a statical one, that of translation, but highly dynamical. A coded order stands not simply for its present contents at its present location, but more fully for any succession of passages of C through it, in connection with any succession of modified contents to be found by C there".

Ein kodierter Befehl steht demnach nicht nur für seinen eigenen Inhalt, sondern auch für jede Folge von Durchgängen von C durch diesen Befehl, wobei sich der Inhalt des Befehls auch ändern kann. Die eng verknüpfte Zusammenarbeit von kodierten Instruktionen und C bei der Steuerung des Geräts, wie sie sich in der „Preliminary Discussion" zeigt (vgl. vorangegangener Abschnitt), wird hier deutlich.[71] Ein kodierter Befehl steht auch für die Operationen von C, wenn diese ihn ausführt bzw. seine Ausführung veranlaßt. Die Autoren schließen, daß das Kodieren als ein logisches Problem gesehen werden muß, das einen neuen Zweig der formalen Logik repräsentiert (7. 1, S. 154):

> „Since coding is not a static process of translation, but rather the technique of providing a dynamic background to control the automatic evolution of a meaning, it has to be viewed as a logical problem and one that represents a new branch of formal logics."

Kodieren ist also nicht statisches Übersetzen, sondern soll einen dynamischen Hintergrund – oder wie es in 7. 3 heißt: ein Kontrollschema für einen hochdynamischen Prozeß – zur Verfügung stellen, weshalb es als neuartiges logisches Problem gesehen werden muß. Um ein genaueres Verständnis dieser Feststellung – welche die Autoren nicht näher erläutern – zu ermöglichen, werde ich zunächst versuchen, zu rekonstruieren, was mit der Entwicklung einer Bedeutung (engl. evolution of a

---

71 Die gegenseitige Dependenz von Code und C bei der Steuerung wird auch in den „Planning and Coding Problems deutlich (7.1, S. 152): „This sequence of codes [for the actual problem] will impose the desired sequence of actions on C by the following mechanism: C scans the sequence of codes, and effects the instructions, which they contain, one by one." Einerseits erlegt die Codesequenz C die gewünschten Aktionen auf, andererseits geht die initiale Aktion, das Lesen der Codesequenz, von C aus, und C bewirkt im Anschluß die in der Sequenz enthaltenen Instruktionen.

meaning) gemeint ist. Anschließend stelle ich Überlegungen dazu an, in welchem Sinne die Autoren das Kodieren als ein neuartiges logisches Problem sehen. Dabei wird auch deutlich, worin die Verschiedenheit zwischen mathematischem Text und Codesequenz besteht.

Der Code als das, was durch das Kodieren zur Verfügung gestellt wird – als das Produkt des Kodierens –, wird in dem letzten Zitat als dynamischer Hintergrund gesehen, der die automatische Entwicklung einer Bedeutung kontrolliert. Welche Bedeutung hier gemeint ist, läßt sich aus den anderen bereits zitierten Textstellen erschließen. Die im Speicher abgelegte Codesequenz bewirkt, daß die Maschine das in Frage stehende Problem löst (s. o., 7.1, S. 152). Dieses Problem ist in dem „meaningful text" gegeben, der in der ersten Phase in der „Sprache der Mathematik" geschrieben wird (s. o., 7.1, S. 152). Die im Speicher abgelegte Codesequenz stellt so gesehen den dynamischen Hintergrund dar, vor dem sich die Bedeutung des mathematischen Textes während des Maschinenlaufs entwickeln kann. Der aktuelle Code für das Problem soll, einfacher ausgedrückt, bewirken, daß die Rechenmaschine das berechnet, was in der mathematischen Problembeschreibung spezifiziert ist.[72]

Deutlich wird hier auch, worin die Verschiedenheit zwischen mathematischer Problembeschreibung und einer zugehörigen Codesequenz in den Augen der Autoren besteht und warum sie entsprechend die mathematische Vorbereitung, die sich aus der Notwendigkeit des zahlenmäßigen Berechnens allgemein ergibt, von den folgenden durch die Maschine bedingten Phasen so deutlich trennen. Das in der ersten Phase erstellte numerische Verfahren und eine zugehörige Codesequenz unterscheiden sich darin, daß die Befehle des Codes nicht nur für ihren eigenen Inhalt stehen, wie es oben heißt (s. o., 7.1, S. 153), sondern auch für die Durchläufe von C durch diesen Befehl. Um einen Befehl und seine Wirkung vollständig erfassen zu können, müssen neben dem Inhalt des Befehls auch alle Durchläufe von C und eventuell damit verbundene Modifikationen des Inhalts des Befehls mitgedacht werden. Der Befehl ist also nur durch Bezug auf die gesamte Codesequenz und die von ihr bewirkten Operationen von C vollständig erfaßbar. Die hochgradige Verflochtenheit der einzelnen Befehle untereinander wird durch die während des Maschinenlaufs automatisch erfolgenden Operationen von C bewirkt, die ihrerseits durch Befehle verursacht sind und im Falle von Sprungbefehlen zu einem nicht linearen Abarbeiten der Codesequenz führen.

Was bedeutet das für das Kodieren? Beim Kodieren ist zu Beginn die Aufgabe klar: Während des von der zu erstellenden Codesequenz gesteuerten Maschinenlaufs sollen die gewünschten Berechnungen durchgeführt werden. Die erforderliche Codesequenz

---

72 In heutiger Terminologie könnte man interpretieren, daß die Beziehung zwischen der mathematischen Problembeschreibung und dem Code für das Problem nicht auf der syntaktischen Ebene – wie das im Falle der statischen Übersetzbarkeit leicht möglich wäre – sondern auf der semantischen Ebene verortet wird. Im damaligen Diskurs wurden diese Begriffe jedoch nicht verwendet.

muß als lineare Folge von Befehlen aufgeschrieben werden; deren Beziehung erschöpft sich jedoch nicht in ihrem Aufeinanderfolgen, sondern ist durch eine hochgradige Dependenz gegeben, die durch das – während des Maschinenlaufs automatische – Operieren von C hergestellt wird. Dieses Zusammenwirken gilt es beim Kodieren und Aufschreiben eines jeden Befehls mitzudenken. Das Kodieren wird als ein neuartiges logisches Problem gesehen, weil nicht nur eine Folge von Rechenschritten erstellt werden muß. Statt dessen müssen vor dem Ausführen der Berechnung auf der Maschine komplexe gegenseitige Abhängigkeiten bewältigt werden, die durch das automatische Operieren einer Instanz realisiert werden, die erst während des Maschinenlaufs in der gegebenen Codesequenz – gesteuert von dieser Sequenz selbst und in Abhängigkeit von erst während des Maschinenlaufs errechneten Werten – hin- und herspringt.

Vor dem Hintergrund der dargestellten Sicht auf das Kodieren wird das Einführen von Flußdiagrammen als Zwischenglied zwischen der mathematischen Problembeschreibung und dem zu erstellenden Code verständlich. Flußdiagramme ermöglichen, wie ich im folgenden aufzeige, mit dem zu beginnen, was der mathematische Text vorgibt: bei der gewünschten Berechnung und in diesem Sinne bei der Bedeutung dieses Textes, die im Maschinenlauf entwickelt werden soll. Flußdiagramme stellen damit, wie man heute sagen würde, eine problemorientierte Notationsweise dar.

Da die gewünschte Codesequenz aus dem mathematischen Text nicht durch statisches Übersetzen erhalten werden kann, ist ein zentrales Problem, das die Autoren in den „Planning and Coding Problems" angehen wollen, einfache Methoden zu finden, die erlauben, den mathematischen Text bzw. seine Bedeutung in eine Codesequenz zu überführen, die diese Bedeutung im Maschinenlauf entwickelt (7.1, S. 154): „Our problem is, then, to find simple step-by-step methods, by which these difficulties can be overcome." In diesem Methodenrahmen spielen Flußdiagramme eine zentrale Rolle. Die Autoren führen sie nach einer längeren und komplizierten Argumentation ein:

Da man, so heißt es in 7.3, beim Aufstellen einer Codesequenz für ein Problem nicht so sehr das anfängliche Aussehen dieser Sequenz selbst, sondern vor allem ihr Funktionieren und ständiges Verändern während des Maschinenlaufs im Kopf haben solle, sei es ratsam, zunächst den Verlauf dieses Prozesses zu planen und daraus die eigentliche Codesequenz in einem anschließenden Schritt abzuleiten (7.3, S. 155):

> „It is therefore advisable [...] to plan first the course of the process and the relationship of its successive stages to their changing codes, and to extract from this the original coded sequence as a secondary operation."

Darüber hinaus sei das Grundmerkmal des Funktionierens des Codes zu sehen in dem Lauf, den C durch die kodierte Sequenz nehme, wobei C der Entwicklung des Prozesses, den es kontrolliert – also der Entwicklung der Bedeutung des mathematischen Textes der ersten Phase – parallel laufe (7.3, S. 155):

„Furthermore it seems equally clear, that the basic feature of the functioning of the code in conjunction with the evolution of the process that it controls, is to be seen in the course which the control C takes through the coded sequence, paralleling the evolution of that process."

Deshalb soll zunächst ein Schema des Laufs von C durch die kodierte Sequenz, genauer: die entsprechende Region im Speicher, angelegt werden, das von ihnen „Flußdiagramm von C" genannt wird (7.3, S. 155):

„We therefore propose to begin the planning of a coded sequence by laying out a schematic of the course of C through that sequence, i. e. through the required region of the Selectron memory. This schematic is the *flow diagram of C*."

Ein Flußdiagramm soll also das „Fließen" von C durch die Speicherregion, die für die Codesequenz benötigt wird, darstellen. Der erste Schritt des Kodierens wird damit eng an die Maschinenfunktion gebunden. Bevor der zum Problem gehörige Code erstellt wird, soll die von ihm unmittelbar gesteuerte Maschinenfunktion, die Operationen von C, geplant werden. Da sich während des Laufs von C die Bedeutung entwickeln soll, die in dem mathematischen Text der ersten Phase enthalten ist, liegt der durch die Codesequenz zu steuernde Lauf von C näher an der mathematischen Problembeschreibung als die diesen Lauf steuernde Codesequenz selbst. Letztere soll entsprechend erst in einem zweiten Schritt – im Rahmen der dritten und vierten Phase der Kodierung – erstellt werden.

In der dritten – der statischen oder mikroskopischen – und der vierten Phase wird das Flußdiagramm in eine Codesequenz übersetzt. In der dritten Phase werden „operation boxes", die Transitionen im Speicher markieren, und „alternative boxes", die eine bedingte Verzweigung des Kontrollflusses abbilden, in Code übersetzt. In der vierten Phase werden alle Speicherpositionen und die Befehlsnummern zugewiesen. Weder das Erstellen der Flußdiagramme noch die dritte und vierte Phase stellen den Autoren zufolge eine kodierende Person vor große Schwierigkeiten. Zur Begründung greifen sie auf ihre Erfahrung und Überzeugung zurück (7.9, S. 171):

„It has been our invariable experience, that once the problem has been understood and prepared in the sense of the preceeding first remark, the drawing of the flow diagram presents little difficulty. Every mathematician, or every moderately mathematically trained person should be able to do this in a routine manner".

Ähnlich heißt es zu der dritten Phase (7.9, S. 172): „We feel certain that a moderate amount of experience with this stage of coding suffices to remove from it all difficulties, and to make it a perfectly routine operation."

Daß das Zeichnen von Flußdiagrammen als auch für „bescheiden mathematisch ausgebildete Personen" einfache Tätigkeit dargestellt wird, verweist auf einen Hintergrund, vor dem das Kodieren als einfach verstanden werden kann, und führt zu dem mathematischen Diskurs, so wie er in der Konzeption des Kodierens in den „Planning and Coding Problems" eine Rolle spielt. Es werden in obigem Zitat mathematisch trainierte Personen als Kodierer anvisiert. Das paßt zunächst dazu, daß

den Codewörtern und Symbolen des Codes – wie in den vorangegangenen Abschnitten aufgezeigt – entsprechend ihrer Aufgabe, die maschinelle Berechnung zu steuern, eine „mathematische Bedeutung“ zugesprochen wird. Das Kodieren, zu dem auch das Zeichnen der Flußdiagramme gerechnet wird, wird entsprechend als eine für mathematisch Ausgebildete besonders leicht zugängliche Tätigkeit gesehen. Übernommen wurde sie in der Anfangszeit der Programmierung häufig von den „Computern“ – typischerweise Frauen –, die zuvor in Rechenbüros mit Hilfe von Tischrechnern die Berechnungen ausführten. Während der „planner“, der zuvor die mathematische Problembeschreibung bzw. das numerische Verfahren erstellte und an ein Rechenbüro weitergab, seine Tätigkeit bei dem in den „Planning and Coding Problems“ vorgestellten Phasenmodell der Kodierung beibehalten konnte, wurden die ehemaligen „computer“ durch die Rechenmaschine ersetzt und in Folge häufig für die Kodierung der Maschine eingesetzt (vgl. dazu Hoffmann 1987, S. 88ff.). Wie zuvor erhielten sie Teile der mathematischen Problembeschreibung der ersten Phase; jedoch rechneten sie nicht mehr selbst, sondern setzten diese Beschreibung in Code um. Wie die Arbeit der Rechnerinnen in den Rechenbüros auf der Liste der mathematischen Tätigkeiten eher unten rangierte, wurde das Kodieren als eine einfache, von „bescheiden mathematisch trainierten Personen“ leicht durchführbare Tätigkeit eingeschätzt. Insofern das Kodieren als Fortsetzung der etablierten Rechenpraxis gesehen wurde, stellt es sich als einfache Tätigkeit dar.

Während aus der Perspektive der Angewandten Mathematik das Kodieren – zumindest mit den vorgestellten Methoden – als einfach erscheint, wurde in dem Text auch deutlich, daß das Kodieren ein schwieriges, sich nicht in einfachem Übersetzen erschöpfendes Problem darstellt. Die Komplexität des Kodierens entfaltet sich dabei eher aus der Perspektive des mathematischen Forschers: Das Kodieren wird in dieser Sicht als logisches Problem gesehen, das einen neuen Zweig der formalen Logik repräsentiert. Als Gegenstand der mathematisch-logischen Wissenschaft erscheint es als schwieriges, neuartiges Problemfeld.

Während einerseits das Kodieren als eng an die Maschine gebundene Tätigkeit konzipiert wird, bei der zunächst der Lauf von C durch den Speicher geplant wird, ermöglicht die wissenschaftlich-mathematische Perspektive, von der Maschine beim Nachdenken über das Kodieren weitgehend zu abstrahieren. Als Problem, das einen neuen Zweig der formalen Logik darstellt, erscheint das Kodieren als ein eigenständiges, von der Maschine in wesentlichen Teilen abtrennbares Wissensgebiet – mit dem Kodieren wird eine eigene Philosophie verbunden (vgl. Preface), und diskutieren wollen die Autoren „the nature of coding per se“ (7.1), also das Wesen des Kodierens als eine Tätigkeit, die für sich stehen kann.[73] In den Aussagen über das Kodieren wird

---

73 Im dritten Teil der „Planning and Coding Problems“ wird Kodieren entsprechend als Tätigkeit dargestellt, die unabhängig von einer bestimmten Maschine gelernt werden kann (12.1, S. 286):

jedoch deutlich, daß es ohne eine Maschine, die einen Speicher und eine Steuereinheit C besitzt, nicht denkbar ist. In der Rede über das Kodieren als eigenständiges, der formalen Logik zuzuordnendes Wissensgebiet wird eine, in einer bestimmten Weise aufgebaute Maschine wie selbstverständlich vorausgesetzt. Das paßt dazu, daß in dem „First Draft“ und eingeschränkt auch in der „Preliminary Discussion“ der generelle Aufbau der Maschine als eine Struktur gedacht wird, die dem Ausführen von Instruktionen zur Durchführung von Berechnungen allgemein unterliegt. Die Rede über das Kodieren als eigenständiges Wissensgebiet setzt diese selbstverständlich erscheinende Struktur ebenso selbstverständlich voraus. Die Maschine wird aus der wissenschaftlich-mathematischen Perspektive als Voraussetzung des Kodierens tendenziell unsichtbar.

Das Kodieren erscheint also, die Ausführungen dieses Abschnittes zusammengefaßt, als eine technisch-mathematische Tätigkeit, wird aber als für mathematisch Ausgebildete prädestiniertes Arbeitsfeld und auch als Gegenstand der mathematischen Forschung konzipiert. Die zu erstellende Codesequenz ergibt sich dem Text zufolge nicht durch statisches Übersetzen aus der mathematischen Problembeschreibung. Der Zusammenhang zwischen mathematischer Problembeschreibung und Code wird deshalb auf einer prozessualen Ebene verortet: die Codesequenz soll die Maschine so steuern, daß die in der mathematischen Problembeschreibung enthaltene Bedeutung entwickelt wird, d. h. die gewünschte Berechnung ausgeführt wird. Das Kodieren setzt in den „Planning and Coding Problems“ an dieser Stelle an: Geplant wird zunächst der Lauf der Maschine – genauer der Lauf von C durch die von der Codesequenz benötigte Speicherregion –, während dem die Bedeutung des mathematischen Textes entwickelt wird. Dieser Lauf wird in einem Flußdiagramm dargestellt.

Als Vorbereitung des Kodierens soll in der Phase der mathematischen Problembeschreibung zunächst ein mathematischer Text erstellt werden, wie das auch bei der Durchführung einer Berechnung durch einen Menschen oder ein Rechenbüro der Fall ist. Es liegt von daher nahe, das Kodieren als Arbeit der in den Rechenbüros arbeitenden Rechnerinnen und Rechner zu sehen, die nun auf Grundlage des mathematischen Textes kodierten statt selbst zu rechnen. Die Codewörter scheinen eine mathematische Bedeutung zu haben, wobei nicht elaboriert wird, was „mathematisch“ hier genauer bedeutet – es wird also auf ein intuitives Mathematikverständnis rekurriert. Als Arbeit mit etwas, das eine mathematische Bedeutung hat, und vor dem Hintergrund der etablierten Arbeitspraxis erscheint das Kodieren als mathematische Tätigkeit. Soweit beim Zeichnen der Flußdiagramme der zur Bedeutung des

---

„It is, of course, justified and even necessary from a didactical point of view, to treat such partial problems, problem fragments – especially in the earlier stages of the instruction of the use of a code, or of coding per se.“

mathematischen Textes gehörige Maschinenlauf und die Speicherplatzvergabe geplant werden muß, erscheint das Kodieren als technische, eng an die Maschine gebundene Tätigkeit. Das Kodieren erfordert also mathematische wie technische Kompetenzen, wird jedoch in den „Planning and Coding Problems“ dem Aufgabengebiet von Mathematikern oder mathematisch trainierten Personen zugerechnet. Das Kodieren erscheint so als Tätigkeitsfeld im Rahmen der Angewandten Mathematik.

Betrachtet als Gegenstand der Forschung wird das Kodieren dem Aufgabenbereich der mathematischen Wissenschaften zugeordnet. Wegen der hochgradigen Interdependenz der – beim Kodieren als lineare Folge zu erstellenden – Befehle einer Codesequenz, die sich erst während des Maschinenlaufs entfaltet, repräsentiert das Kodieren aus einer mathematisch-wissenschaftlichen Perspektive ein Problemfeld, das als ein neuer Zweig der mathematischen Logik gesehen wird.

2.5 Zusammenfassung: Die Speicherprogrammierung zwischen technischem Mechanismus und mathematischer Anweisung

In diesem Abschnitt werte ich die Ergebnisse der drei in diesem Kapitel interpretierten Texte im Zusammenhang aus. Dabei stelle ich zunächst überblicksartig den Einfluß technischer, neurophysiologischer und mathematischer Diskurse auf die Konzeption der Rechenmaschine, des Codes sowie der Kodierung zusammen, so wie sie in diesen Texten deutlich werden. Anschließend arbeite ich heraus, wie sich vor diesem Hintergrund die Hybdridobjekt-Sichtweise auf den Code und die Rechenmaschine konstituiert.

Vor dem Hintergrund technischer Diskurse stellt sich das „automatic computing system“, das „device“ bzw. das „electronic computing instrument“ als aus technischen Bauteilen aufgebaute Maschine dar. Der Code, die Codewörter bzw. die Instruktionen oder Befehle erscheinen als im Speicher abgelegte, technisch realisierte Artefakte, deren Verwandtschaft mit technischen Bauteilen thematisiert wird. Sie scheinen allein oder in engem Zusammenwirken mit der zentralen Steuereinheit C die Rechenmaschine zu steuern, zu kontrollieren oder zu instruieren und stehen nicht nur für ihren eigenen Inhalt, sondern auch für die Durchläufe von C durch den Befehl. Der Aufbau der Maschine und des Codes wird entsprechend ihrer engen Verwobenheit in gegenseitiger Abhängigkeit entwickelt. Das Kodieren wird vor diesem Hintergrund als eng an die Maschine gebundene Arbeit deutlich; geplant werden muß z. B. der Lauf von C durch die Speicherregion, in der der Code abgelegt wird, sowie die Speicherplatzvergabe.

Der Einfluß neurophysiologischer Diskurse drückt sich darin aus, daß das Rechengerät als nervensystemähnlicher Organismus gezeichnet wird. Entsprechend der Organ- und Neuronenmetaphorik, mit der es beschrieben wird, erscheint es als spürend, kommunizierend, verstehend, selbständig ausführend und rechnend. Der Code erscheint entsprechend als Liste von Instruktionen und Befehlen, die verstanden und ausgeführt werden, und auch als Sprache, die die Maschine versteht. Das Kodieren besteht so gesehen in dem Instruieren bzw. Befehlen eines elektronischen Nervensystems und Gehirns oder, wie es einmal heißt, als Aufbereiten eines Problems in der Sprache, die die Maschine versteht.

Um den Einfluß mathematischer Diskurse zusammenzufassen, möchte ich hier unterscheiden zwischen angewandter und forschender Mathematik. Mehrtens charakterisiert die Differenz zwischen diesen beiden wie folgt (Mehrtens 1998, S. 177):

> „Wir haben mit der mehrfachen Bedeutung des Wortes ‚Mathematik' umzugehen: Mathematik ist die Technik des Umgangs mit speziellen Symbolsystemen wie denen der elementaren Arithmetik. Mathematik ist zugleich die von den verbreiteten Kulturtechniken abgesonderte wissenschaftliche Kultur der konstruktiv forschenden Arbeit an derartigen Symbolsystemen."

Da die Rechenmaschine die etablierte Rechenpraxis verbessern und das Ausführen von Berechnungen beschleunigen sollte, wird sie in den besprochenen Texten zum einen aus der angewandten Mathematik heraus verstanden und erscheint in dieser Hinsicht als maschinelle Technik zum Umgang mit Symbolsystemen. Zum anderen wurden die Rechenmaschine und der Code auch als Gegenstände der mathematisch-logisch forschenden Wissenschaft gesehen und in dieser Hinsicht selbst als Symbolsysteme konzipiert.

Zunächst zum Diskurs der Angewandten Mathematik: Die Maschine wird in den besprochenen Texten als „general purpose computing machine" oder als elektronisches Recheninstrument gesehen, das Instruktionen zwecks Durchführung von Berechnungen ausführen soll. Damit scheint sie die Rechnerinnen, die bis dahin in Rechenbüros die Berechnungen durchführten – diese Arbeit wurde zumeist von Frauen ausgeführt –, weitgehend zu ersetzen. Da den Codewörtern eine mathematische Bedeutung zugeschrieben wird, wirken die Instruktionen des Codes wie Anweisungen mit einer mathematischen Bedeutung, so wie sie auch an eine menschliche Rechnerin gerichtet sein können. Der Code rückt so gesehen nah an die „Sprache der Mathematik" heran, zumal auch mathematische Texte oft auffordernden oder befehlenden Charakter haben. So stellt Mehrtens mit Bezug auf Rotman fest (Mehrtens 1990, S. 442f.):

> „Die Verben dieser [der mathematischen] Sprache stehen vor allem im Imperativ (Rotman 1988, S. 8): ‚Mathematics is so permeated by instructions for actions to be carried out, orders, commands, injunctions to be obeyed [...] that mathematical texts seem at time to be little more than sequences of instructions written in an entirely operational, exhortatory language.' "

Das Kodieren beginnt aus der Sicht der damals etablierten Rechenpraxis in etwa dort, wo zuvor das Berechnen in den Rechenbüros ansetzte: nach einer vorbereitenden Phase der mathematischen Problembeschreibung. Es ist so naheliegend, das Kodieren als Arbeit der in diesen Rechenbüros arbeitenden Rechnerinnen zu sehen. Statt selbst die Berechnung durchzuführen, kodierten sie (bzw. einige von ihnen) nun die Rechenmaschine. Wie die Arbeit der Rechnerinnen galt auch das Kodieren aus dieser Sicht als eher einfache Tätigkeit.

Um einen Bereich auch als Gegenstand der mathematischen Forschung begreifen zu können, muß er zunächst – folgt man obigem Zitat (Mehrtens 1998, S. 177) – als Symbolsystem konzipiert werden. Die Rechenmaschine wird aus der Perspektive des mathematischen Forschers im „First Draft" zunächst als Struktur aus hypothetischen Elementen vorgestellt, die dem Ausführen von Instruktionen zur Durchführung von Berechnungen allgemein zu unterliegen scheint. So gesehen können weite Teile des Rechneraufbaus ohne Rekurs auf maschinell-technische Details beschrieben werden. Der Weg ist geöffnet, um das logische Design der Maschine zu betrachten, wie es in der „Preliminary Discussion" heißt. Rechnerbau erscheint so auch als mathematisch-logisches Problem und damit als Gegenstand der mathematischen Wissenschaft. Der Code erscheint in dem „First Draft" – da ausgeführt von einem Gerät, dessen Struktur dem Ausführen von Instruktionen zur Durchführung von Berechnungen allgemein unterliegt – implizit als Liste von Befehlen, die zur Durchführung von Berechnungen benötigt werden. So gesehen stellt der Aufbau des Codes weniger ein technisches als ein mathematisches Problem dar. In den „Planning and Coding Problems" wird auch das Kodieren als logisches Problem gesehen, das einen neuartigen Zweig der formalen Logik repräsentiert, und es erscheint als mehr oder weniger eigenständiges Wissensgebiet.

Einerseits erscheint das Kodieren also als eigenständiges, von dem Aufbau einer speziellen Maschine unabhängiges Wissensgebiet, andererseits ist wichtiger Teil des Kodierens, den Lauf von C durch den Speicher zu planen – und damit eine Maschinenfunktion, die abhängig ist von dem speziellen Aufbau der Maschine. Daß dies nicht als widersprüchlich wahrgenommen wurde, dokumentiert den Einfluß des neurophysiologischen Diskurses auf die besprochenen Papiere: Vor dem Hintergrund des neurophysiologischen Diskurses erscheint die Rechenmaschine als Äquivalent des Nervensystems. Nervensystem und Gehirn scheinen prinzipiell gleichartig zu funktionieren und im wesentlichen die gleiche Struktur materiell zu realisieren. Bei der Konstitution des Kodierens als eigenständigen Wissensgebiets wird diese scheinbar allgemeingültige Struktur selbstverständlich mitgedacht. Das Kodieren kann aus der mathematischen Forscherperspektive als rein symbolische Tätigkeit und neuer Zweig der formal-logischen Forschung gedacht werden, weil die Spezifika der Maschine vor dem Hintergrund des neurophysiologischen Diskurses unsichtbar werden.

Die Rechenmaschine und der Code werden also erstens als technische Artefakte begriffen. Vor dem Hintergrund neurophysiologischer Diskussionen der Zeit war es möglich, die Maschine zweitens in Analogie zum menschlichen Nervensystem und Gehirn zu verstehen und sie als eine Art Organismus darzustellen. Der Code wird entsprechend als Liste von Befehlen gedacht, welche die Maschine versteht und ausführt. Da die Maschine – gedacht als Nervensystem bzw. Gehirn – im wesentlichen so wie ein Mensch zu rechnen scheint, wirkt sie wie ein Ersatz des menschlichen Rechners, der Code – zumal mit mathematischer Bedeutung versehen – als eine Art Äquivalent zu mathematischen Anweisungen. Die Analogie Rechenmaschine-Nervensystem erlaubt drittens auch, zu abstrahieren und zu verallgemeinern: Das Gerät des „First Draft" erscheint auch als Struktur, die dem Ausführen von Instruktionen zur Durchführung von Berechnungen allgemein unterliegt, der Code als Liste von Befehlen zur Durchführung von Berechnungen. Die Konstruktion der als Symbolsystem konzipierten Rechenmaschine und ihres Codes kann so auch als Gegenstand der mathematisch-wissenschaftlichen Forschung gesehen werden.

In den „Planning and Coding Problems" widersprechen von Neumann und Goldstine nach einiger Erfahrung mit dem Kodieren explizit der – folglich virulenten – Annahme, daß mathematische Anweisungen und Code eine Art Äquivalent darstellen: Eine Codesequenz zur Steuerung einer Berechnung sei nicht durch einfaches Übersetzen aus der Sprache der Mathematik in die Sprache des Codes zu erhalten. Denn ein kodierter Befehl stehe nicht nur für seinen Inhalt, sondern auch für die Durchläufe von C durch ihn. Ein Befehl kann demnach nur in engem Zusammenhang mit der Maschinenfunktion verstanden werden. Der im Speicher abgelegte Code stellt einen „dynamischen Hintergrund für die Entwicklung einer Bedeutung" bereit – der Begriff Maschine ist jedoch in diesem Zusammenhang in dem Text nicht zu finden.

Es ist an dieser Stelle ein prinzipieller Unterschied zwischen einem mathematischen Text, in dem z. B. ein numerisches Verfahren dargestellt ist, und einer zugehörigen Codesequenz angesprochen: die enge wechselseitige Abhängigkeit zwischen Code und Maschine wird sichtbar; die Anweisungen an die rechnende Maschine und den rechnenden Menschen scheinen sich grundlegend zu unterscheiden. Möglich wäre hier, das Kodieren als eine technische, eng an die Maschine gebundene Tätigkeit zu konzipieren – es wird jedoch als neuartiges logisches Problem betrachtet und erscheint als in wesentlichen Teilen eigenständiges Wissensgebiet. Deutlich wird hier, daß in den besprochenen Texten – obwohl der Einfluß technischer Diskurse unübersehbar ist – eine mathematisch-logische Forscherperspektive auf Probleme des Rechenmaschinenbaus, des Codes und des Kodierens zentral ist. Gestützt auf Deutungen, die Ausdruck des neurophysiologischen Diskurses sind, ermöglicht diese Perspektive, den Code und die Rechenmaschine als Realisierung einer prinzipiell allgemeingültigen Struktur zu begreifen, und sie als Symbolsysteme mit Bedeutung auch für den Bereich

des menschlichen Rechnens zu fassen: die Rechenmaschine als Struktur aus hypothetischen, neuronengleichen Elementen, den Code als Liste von Befehlen mit Ähnlichkeit zur mathematischen Sprache. Die Rechenmaschine und der Code sind einerseits materiell-technische Gegenstände, andererseits werden sie auch als Symbolsysteme mit Bedeutung für den natürlich/biologischen Bereich konzipiert. Sie erscheinen als Hybridobjekte, die als technisch, als natürlich und als Zeichensystem gedacht werden können.

Kapitel 3: Immaterielle Maschinen

In dem „First Draft of a Report on the EDVAC“ führte von Neumann, wie in Abschnitt 2.2.2 aufgezeigt, zur Darstellung seiner Überlegungen in Anlehnung an die Neuronenmodelle von McCulloch und Pitts eine über den konkreten technischen Gegebenheiten angesiedelte Abstraktionsebene ein – ein Umstand, der ermöglichte, auf einer allgemeineren, die Maschine transzendierenden Ebene zu diskutieren und Rechnerbau auch als mathematisch-logisches Problem zu begreifen. So betrachtet reiht sich der „First Draft“ in eine Serie von Arbeiten ein, in denen von Neumann mathematische Vorgehensweisen auf dafür nicht prädestiniert scheinende Bereiche anwendete. Dazu Burks (1987b, S. XIII):

> „Much of von Neumanns work is characterized by the application of mathematical models to the foundation of disciplines that a priori do not seem amenable to such treatment. This is true of his work in quantum mechanics, the theory of games, economic behaviour, and the theory of automata.“

Die in dem „First Draft“ eingeführte Abstraktionsebene idealisierter Neuronenmodelle legte den Grundstein für eine solche, zunächst unerwartete Anwendung mathematischer Sichtweisen: den mit der Automatentheorie verfolgten Versuch, mathematische Modelle zur Verfügung zu stellen, die es erlauben sollten, Maschinen und Organismen – und besonders Rechengeräte und das Gehirn bzw. Nervensystem – einheitlich beschreiben und damit vergleichen zu können. Der „First Draft“ kann als Anfang von von Neumanns Arbeiten zur Automatentheorie gesehen werden.

Der Vergleich zwischen lebenden Organismen – besonders dem seiner Meinung nach kompliziertesten Organismus, dem menschlichen Zentralnervensystem – und künstlichen Automaten, insbesondere Rechenmaschinen, sollte von Neumann bis zu seinem Tod 1957 beschäftigen. Besonders bekannt geworden sind seine diesbezüglichen Überlegungen in „The General and Logical Theory of Automata“, einem Vortrag auf dem Hixon-Symposium im September 1948, den ich im folgenden genauer betrachten will. Dieser Vortrag war der erste, in dem er eine Theorie der Automaten einforderte und in ihren Grundzügen skizzierte. In der Folgezeit arbeitete von Neumann die hier angerissenen Ideen im Rahmen mehrer Vorträge bzw. Vortragsreihen und dazugehöriger Manuskripte, die zumeist von anderen in eine veröffentlichungsfähige Textform gegossen wurden, weiter aus. Im Dezember 1949 hielt er an der Universität von Illinois eine Vortragsreihe „Theory and Organization of Complicated Automata“, die von Burks rekonstruiert wurde (von Neumann 1949). Im Rahmen eines Vortrags in Princeton 1948 sowie in einem 1952 und 1953 geschrieben Manuskript, das von Burks nach von Neumanns Tod vervollständigt wurde, detaillierte er seine Ideen zu sich selbst reproduzierenden Automaten (von Neumann 1953). Im Januar 1952 hielt er fünf weitere Vorträge am „California Institute of Technology“, die von R. S. Pierce mitgeschrieben wurden und in denen er besonders seine Gedanken zur

Fehlerbehandlung darlegte (von Neumann 1952).[74] Seine letzte Arbeit war „The Computer and the Brain", ein unvollendetes Manuskript für die Silliman Lectures, in dem er den Vergleich zwischen Rechenmaschinen und menschlichem Gehirn und Nervensystem zusammenfaßte (von Neumann 1958).

Bis heute gilt die Arbeit an einer Theorie der Automaten als nicht abgeschlossen; eine befriedigende Theorie scheint hier noch auszustehen. Von Neumanns diesbezügliche Überlegungen erscheinen als bis in die neuere Zeit aktuell. So schreibt Mahr (1984, S. 101):

> „Die Arbeit an einer Theorie der Automaten und der Information ist vorangeschritten, hat aber nicht überall die in sie gesetzten Hoffnungen erfüllt. Immer noch scheint es, als seien wir erst am Anfang, und immer noch sehen wir in einer sehr fortgeschrittenen und subtilen Theorie der Automaten die Grundvoraussetzungen dafür, die zur Verfügung stehenden technischen Möglichkeiten zu nutzen."

Ähnlich hält auch Burks fest, daß es zwar viele Versuche gegeben habe, eine mathematische Theorie der Art, wie sie von Neumann vorschwebte, zu entwickeln, jedoch sei dieses Ziel nicht erreicht worden. Dabei ist besonders seine Einordnung der Automatentheorie in spätere Forschungsgebiete interessant und mag einen ersten Eindruck von der Breite des Gebiets geben, auf das eine Theorie der Automaten anwendbar sein sollte (Burks 1987d, S. 363):

> „Such a theory has not been achieved. Others have proposed similiar or related subjects: cybernetics, control theory, learning theory, logic of discovery, artificial intelligence, intelligent systems, machine intelligence, robotics, self-organizing systems, self-repairing systems, theory of adaptive systems, mathematical theory of evolution, and, most recently, cognitive science. Although much of value has been accomplished in these areas, there is still no general mathematical theory of the kind von Neumann envisaged. One might hold that, when automata become highly complex, there is no theory, only engineering, but von Neumann clearly thought otherwise."

Im letzten Kapitel habe ich anhand der Texte, die von Neumann allein oder mit Goldstine und Burks zum Thema speicherprogrammierbare Rechenmaschinen und deren Kodierung verfaßte, drei verschiedene Perspektiven auf das Rechengerät und den Code herausgearbeitet:

---

74 Daß von Neumanns Überlegungen damals große Teile der mathematischen Community befremdeten, wird deutlich in den Erinnerungen von J. Hartmanis (1979, S. 225): „During this period I met for the first time Johnny von Neumann when he lectured in 1952 at Cal Tech on his work on ‚Probablistic Logics and the Synthesis of Reliable Organisms from Unreliable Components'. I must admit that at this time I failed completely to understand the importance of von Neumann's work on computing machines and automata theory. His enthusiasm and brilliance were obvious, but my indoctrination in pure mathematics was too strong to be penetrated by such ideas at this time. As a matter of fact, many of the Cal Tech mathematicians lamented the fact that ‚Johnny had given up to do real mathematics.' "

- die technische Perspektive, aus der das Gerät als Maschine und die Codewörter als im Speicher technisch realisierte Artefakte betrachtet werden, die die Maschine steuern;
- eine neurophysiologisch geprägte Sicht, aus der das Gerät als nervensystemartiger Organismus erscheint, der ähnlich wie ein menschlicher Rechner Instruktionen mit mathematischer Bedeutung und Sprachcharakter erhält, in seinem Gedächtnis ablegt, versteht, und eine entsprechende Rechnung ausführt;
- eine mathematische Forscherperspektive, in der Rechengerät und Code als Symbolsysteme konzipiert werden.

Dieses Oszillieren der Perspektiven, in dem sich die Hybridobjekt-Sichtweise dokumentiert, spielt auch in der in diesem Kapitel im Mittelpunkt stehenden „General and Logical Theory of Automata" eine entscheidende Rolle: Automaten können hier Maschinen und Organismen gleichermaßen sein und werden, ähnlich wie das Rechengerät des „First Draft", auch als Struktur aus hypothetischen Elementen gedacht.

Da in dem Papier „The General and Logical Theory of Automata" neben Rechenmaschinen auch die Natur und natürliche Organismen ausführlicher thematisiert werden, wird in ihm der Orientierungsrahmen deutlicher, vor dessen Hintergrund von Neumann seine Ideen entwickelte. Prägnanter noch als in den im letzten Kapitel besprochenen Texten dokumentiert sich, daß zwei Dualismen – erstens der Dualismus zwischen natürlich und technisch und zweitens der zwischen dem technisch gedachten Organismus und einem Symbolsystem (und in diesem Sinne zwischen materiell und immateriell) – in Frage gestellt und neu gedacht werden.

In diesem Kapitel wird es also zum einen darum gehen, den Orientierungsrahmen zu rekonstruieren, der erlaubt, technisch und natürlich und vor allem handelnde körperliche Entitäten und Symbolsysteme sehr nah aneinander zu denken. Dazu analysiere ich in Abschnitt 3.1 das Papier „The General and Logical Theory of Automata" in Hinblick darauf, wie Organismen und ihre Beschreibung miteinander in Beziehung gesetzt werden. Ich arbeite heraus, daß eine Kombination aus atomistischen und behavioristischen Sichtweisen dabei eine zentrale Rolle spielt: In einer atomistischen Sicht lassen sich natürliche wie technische Entitäten in klar voneinander isolierbare Einheiten unterteilen. Diese werden als „black boxes" und Stimulus-Response-Automatismen betrachtet, deren Verhalten als eindeutig beschreibbar erscheint. Körperliche Entitäten können aus dieser Perspektive als Verbindungsmuster von hypothetischen Ein-/Ausgabemechanismen – als agierende Symbolsysteme – gesehen werden. Sie werden entsprechend auch als ihre eigene Beschreibung gedacht. Während einerseits Körper als Beschreibung vorgestellt werden, erscheinen andererseits Beschreibungen als körperliche Entitäten. In Anlehnung an die damalige Vorstellung vom Gen werden Beschreibungen und Instruktionen als materielle und

steuernde Entitäten beschrieben. Technisch konzipierte Organismen erscheinen als Symbolsysteme und umgekehrt Symbolsysteme als handelnde, materielle Einheiten.

Vor dem Hintergrund, daß Beschreibungen und Instruktionen auch als materielle, handelnde Entitäten vorgestellt werden, sind Beschreiben, Befehlen und materielles Konstruieren kaum zu unterscheiden. Das führt zu Abschnitt 3.2, der insofern etwas aus dem Rahmen fällt, als ich hier die Zeit um 1948 verlasse und die spätere Entwicklung der Programmierung thematisiere: Ich reflektiere, was die von Neumannschen Überlegungen zu Automaten zur Charakterisierung des Programmierens bis heute beitragen können. Dazu beleuchte ich den Zusammenhang zwischen Genetik und Programmierung genauer (3.2.1): Der Import der Kodier- und Programmiermetaphorik in die Genetik erlaubte, Programme in ein Sinngeflecht einzubetten, in dem sie die Rolle der Rekonstruktion einer Wirklichkeit spielen können, die selbst als eine Art Programm erscheint. In Abschnitt 3.2.2 gehe ich auf die Rolle behavioristischer und atomistischer Sichtweisen in der Programmierung ein und stelle Überlegungen dazu an, wie dieses Sichtweisen dazu beitragen, daß im Diskurs der Programmierung Mensch, Maschine und Symbolsystem eng aneinander gedacht werden.

In Abschnitt 3.3 betrachte ich den Teil meiner Arbeit, der sich mit dem Wirken von Neumanns auseinandersetzt (Kapitel 2 und 3) im Zusammenhang und spüre der Rolle nach, die ein behavioristischer Handlungsbegriff für die technische Sicht auf den menschlichen Organismus, die Anthropomorphisierung der Rechenmaschine und die Konzipierung beider als Symbolsysteme hat. Abschließend verfolge ich den Einfluß, den von Neumanns Arbeiten auf die Entstehung der Informatik als wissenschaftliche Disziplin hatten. Hier vertrete ich die Ansicht, daß die Analogie Rechenmaschine-Nervensystem – und damit das Sinngeflecht, in das die Rechenmaschine in den von Neumannschen Texten eingebettet wird – in der Konstitution dieser Wissenschaft eine zentrale Rolle spielt.

3.1 Text: „The General and Logical Theory of Automata“

Der Vortrag „The General and Logical Theory of Automata“, den von Neumann im September 1948 auf dem Hixon Symposium hielt, kreist um die Frage, warum die Community von Wissenschaftlern, zu der auch von Neumann sich zählt („we“), noch nicht in der Lage ist, Automaten zu bauen, die so komplex sind wie natürliche Organismen. In dem Vortrag entwickelt von Neumann anhand des Vergleichs mit der Natur Forschungsthemen und Fragen, die seiner Meinung nach vor Erreichung dieses Ziels bearbeitet werden müssen. Dabei erscheint die Natur als eine Art Baumeisterin, die die Organismen nach rationalen Prinzipien geschöpft hat. Z. B. wird die Frage diskutiert, warum die Natur die „count method“ der „decimal-“ oder „binary-expansion method“

zur Übertragung von taktilen Reizen über die Nervenbahnen an das Gehirn vorziehe und von Neumann kommt zu dem Schluß (von Neumann 1948, S. 308): „In the case under discussion, nature has obviously resorted to an even more redundant and even safer system." Organismen scheinen entsprechend diesem Bild einer vernünftig entscheidenden Natur nach rationalen Prinzipien aufgebaut zu sein.

Einerseits soll die Natur in dem Vortrag also ein Vorbild für den Bau von Automaten liefern, andererseits ist der Blick auf die Natur durch die Rechenmaschine vorstrukturiert. Das Problem der Organismen und der Organisation soll aus dem Blickwinkel der Rechenmaschine analysiert werden (S. 290): „It is nevertheless of a certain interest to analyse the problem of organisms and organization from the point of view of these relatively small, artificial automata". Die moderne, speicherprogrammierbare Rechenmaschine liefert also Folie, durch die Organismen betrachtet werden. Sie erscheint zugleich als Prototyp auf dem Weg zum Bau komplexer und flexibler Automaten (S. 290): „The prototypes for these systems [which function with that type of logical flexibility and autonomy that we find in the natural organisms] are the modern computing machines."

Das doppelte Anliegen, von der Natur für den Bereich der technischen Konstruktion und umgekehrt von dem Bereich der technischen Konstruktion für die Interpretation der Natur lernen zu wollen,[75] drückt sich nicht zuletzt in der verwendeten Begrifflichkeit aus. So werden sowohl der ENIAC als auch das menschliche Nervensystem als „Organismen" (z. B. S. 301) bezeichnet, es wird also biologische Terminologie in den technischen Bereich übertragen. Umgekehrt wird das menschliche Nervensystem als „Automat" oder auch „digitaler Mechanismus" (S. 306) beschrieben, also technische Begrifflichkeit in den Bereich lebendiger Organismen übertragen. Die verwendete Begrifflichkeit läßt damit für die Benennung substantieller Verschiedenheiten zwischen Mensch und Maschine nur wenig Raum. Es dokumentiert sich das Aufweichen des ersten oben angesprochenen Dualismus zwischen natürlich und technisch.

Die hier deutlich werdende technische Konzeption von Organismen ist die Voraussetzung für das Verschwimmen des zweiten Dualismus zwischen Organismus und Symbolsystem, wie ich im folgenden aufzeigen werde. In einem ersten Schritt gehe ich dabei von den Gegenständen aus und rekonstruiere, wie sie in „The General an Logical Theory of Automata" auch als Beschreibung gedacht werden (Abschnitt 3.1.1). In einem zweiten Schritt (Abschnitt 3.1.2) setze ich bei den Beschreibungen an und zeichne nach, daß sie auch als materielle, operierende Instanzen vorgestellt werden.

---

75 In der Einleitung des Papiers formuliert von Neumann dieses Anliegen wie folgt (S. 289): „Nevertheless, some regularities which we observe in the organization of the former [natural organisms] may be quite instructive in our thinking and planning of the latter [artficial automata]; and conversely, a good deal of our experiences and difficulties with our artificial automata can be to some extent projected on our interpretations of natural organisms."

3.1.1 Objekte als Beschreibung

In der „General and Logical Theory of Automata" werden Organismen wie Maschinen als regelhaft funktionierende Systeme *sehr vieler* sie konstituierender Elemente konzipiert. Der Unterschied zwischen biologisch und technisch erscheint dabei vor allem als quantitatives Problem. Möglich ist dies, weil Neuronen als das betrachtet werden, was Vakuumröhren in Rechenmaschinen sind – als „basic switching organs" (S. 299):

> „The basic switching organs of the living organisms, at least to the extent to which we are considering them here, are the neurons. The basic switching organs of the recent types of computing machines are vacuum tubes;"

Dabei scheinen Neuronen und Vakuumröhren nicht nur in bezug auf ihre Funktionalität, sondern auch in bezug auf ihren materiellen Aufbau ähnlich zu sein. So heißt es in dem Kapitel „Comparisons between Computing Machines and Living Organisms" (S. 301):

> „In the vacuum tube the critical area of control is the space between the cathode (where the active agents, the electrons, originate) and the grid (which controls the electron flow). This space is about one millimeter deep. The corresponding entity in a neuron is the wall of the nerve cell, the ‚membrane'. "

Gesehen als Aggregate sehr vieler im Grunde gleich beschaffener Elemente erscheint der Unterschied zwischen modernen Rechenmaschinen und dem zentralen Nervensystem im wesentlichen als Problem der Größenordnung: Die „enorme" Komplexität lebender Organismen besteht dem Papier zufolge in der hohen Zahl ihrer Elemente: 10^{15} bis 10^{16} Zellen für den menschlichen Körper bzw. 10^{10} Neuronen für das zentrale Nervensystem. Während moderne Rechenmaschinen aus maximal 10^{4} „basic switching organs" (gemeint sind Vakuumröhren) aufgebaut sein könnten, bestehe das zentrale Nervensystem aus ca. 10^{10} „basic switching organs" – Neuronen in diesem Fall (vgl. S. 299f.).[76]

Die Konzipierung von Neuronen wie auch Vakuumröhren als „basic switching organs" geht einher mit einer technischen Wahrnehmung des Nervensystems. Wie schon in dem „First Draft" sieht von Neumann auch in „The General and Logical Theory of Automata" in Anlehnung an McCullochs und Pitts Artikel „A Logical Calculus of the Ideas Immanent in Nervous Activity" von 1943 das Nervensystem als gleichmächtig zu einem Mechanismus (S. 309f.):

> „It has often been claimed that the activities and functions of the human nervous system are so complicated that no ordinary mechanism could possibly perform them. [...] The

---

76 Die Auffassung, daß der Unterschied zwischen lebenden Organismen und toter Materie auf Komplexitätsunterschieden beruhe, war auch in der Genetik der Zeit, auf die ich weiter unter noch zu sprechen komme, verbreitet (vgl. z. B. Beadle 1948, S. 71).

McCulloch-Pitts result puts an end to this. It proves that anything that can be exhaustively and unambiguously described, anything that can be completely and unambigously put into words, is ipso facto realizable by a suitable finite neural network. Since the converse statement is obvious, we can therefore say that there is no difference between the possibility of describing a real or imaginated mode of behavior completely and unambiguously in words, and the possibility of realizing it by a finite formal neural network."

Deutlich wird anhand des Zitats auch, daß die Annahme des Nervensystems als Äquivalent eines McCulloch/Pitts-Netzwerk und Mechanismus[77] Folgen dafür hat, was als vollständig und eindeutig beschreibbar gesehen wird. Ist etwas vollständig und eindeutig beschreibbar, so heißt es hier, folgt daraus, daß es als endliches neuronales Netzwerk realisiert werden kann und umgekehrt[78]: Jede Verhaltensform, die das Nervensystem – gedacht als McCulloch/Pitts-Netzwerk – hervorbringt, ist vollständig und eindeutig beschreibbar.

Diese Annahme der vollständigen und eindeutigen Beschreibbarkeit von Verhalten spielt in dem Papier eine zentrale Rolle. In Zusammenhang mit einer atomistischen Betrachtungsweise erlaubt sie, Organismen und Maschinen nicht nur in bezug auf das nach außen beobachtbare Verhalten zu vergleichen. Dadurch, daß Organismen und Maschinen aus vielen voneinander isolierbaren Elementen aufgebaut zu sein scheinen, wird möglich, die Annahme der vollständigen und eindeutigen Beschreibbarkeit des Verhaltens auch auf das Verhalten dieser Elemente anzuwenden, was von Neumann im Rahmen der von ihm so benannten „axiomatic procedure" tat. Diese werde ich nun genauer vorstellen:

Mit der Sicht auf Organismen als Aggregate sehr vieler Elemente öffnete von Neumann einen neuen Problemraum (S. 290): „We have absolutely no past experience with systems of this degree of complexity."[79] Das entstandene neuartige „Problem, das die natürlichen Systeme repräsentieren", unterteilte er zunächst in zwei Teile:

---

77 Pitts und McCulloch behaupteten am Ende ihres Papiers die Äquivalenz organischer, neuronaler Netze mit Turings, Churchs und Kleenes Berechenbarkeitsformalismen (McCulloch, Pitts 1943, S. 25): „If any number can be computed by an organism, it is computable by these definitions and conversely.", den Begriff Mechanismus benutzten sie jedoch nicht. Da McCulloch bei von Neumanns Vortrag 1948 anwesend war und in seinen auf den Vortrag folgenden Diskussionsbeiträgen von Neumanns Interpretation seiner und Pitts Resultate nicht widersprach, gehe ich davon aus, daß er diese im wesentlichen teilte. Die Schlußfolgerungen, die McCulloch und Pitts für ihre formalen Netzwerke, die nach ihnen aus gegenüber echten Neuronen vereinfachten Modellen aufgebaut waren, ableiteten, galten ihnen zufolge genauso für organische Netzwerke (S. 19): „The importance of the formal equivalence lies in this: that the alterations [of the responsivenes of natural neurons] actually underlying facilitation, extinction and learning in no way affect the conclusions which follow from the formal treatment of the activity of nervous nets, and the relations of the corresponding propositions remain those of the logic of propositions."

78 Die umgekehrte Richtung ist nach von Neumann offensichtlich, d. h. es wird vorausgesetzt, daß, wenn etwas gebaut bzw. realisiert ist, das entstandene Produkt auch vollständig und widerspruchsfrei beschrieben werden kann.

79 Auf Grund der hohen Komplexität, mit der in dieser Sicht besonders auch im Falle des Nervensystems umgegangen werden muß, ist von Neumann der Meinung, daß die Entwicklung

1. Die Struktur und das Funktionieren der elementaren Einheiten zu untersuchen sei Aufgabe der organischen und physikalischen Chemie sowie der Quantenmechanik.
2. Die Frage, „how these elements are organized into a whole, and how the functioning of the whole is expressed in terms of its elements", sei hingegen dazu angetan, „those of us who have the background and the tastes of a mathematician or a logician" anzuziehen. Qua „Axiomatisierung" des Verhaltens könnten diese sich des ersten Teils des Problems entledigen: „With this attitude, we will be inclined to remove the first part of the problem by the process of axiomatization". Die Elemente werden dabei als „black boxes" behandelt (S. 289f.):

   „They are viewed as automatisms, the inner structure of which need not be disclosed, but which are assumed to react to certain unambigously defined stimuli, by certain unambiguously defined responses.

   This being understood, we may then investigate the larger organisms that can be built up from these elements, their structure, their functioning, the connections between the elements, and the general theoretical regularities that may be detectable in the complex synthesis of the organisms in question."

Auf Grundlage der Voraussetzung, daß die Elemente als „black boxes", als regelhaft funktionierende Stimulus-Response-Mechanismen beschrieben werden, kann das Verhalten der einzelnen Elemente als mathematische Funktion gefaßt werden. Ob es möglich ist, die Elemente so zu isolieren und als Automatismen eindeutig beschreiben zu können, wird hier nicht problematisiert. Das paßt dazu, daß auch das Verhalten, das das Nervensystem als ganzes hervorbringt, als vollständig und eindeutig beschreibbar vorgestellt wird. Als fraglich erscheint nur, ob die Beschreibung des Verhaltens der einzelnen Elemente auch korrekt ist. Dies festzustellen wird als Aufgabe der ersten Hälfte des Problems gesehen.

Daß in einem zweiten Schritt auch die gesamten Organismen als regelhaft aus diesen „Axiomen" aufgebaute Entitäten gefaßt werden, erlaubt es, die Erforschung von Organismen als Ganzes nicht zuletzt als Aufgabenbereich der Mathematik und Logik zu betrachten. Denn Organismen erscheinen aus dieser Perspektive als Organisationen von Stimulus-Response-Mechanismen, von deren materiellem Aufbau bei der Erklärung ihrer Funktionsweise abstrahiert werden kann. Sie können so gesehen in wesentlichen Aspekten erforscht und beschrieben werden, ohne daß auf ihre spezielle Materialität und die mit dieser befaßten Wissenschaften rekurriert werden muß, wie das schon im „First Draft" durch die Einführung hypothetischer Elemente für die Beschreibung der Rechenmaschine erreicht wird. Dem materiellen Aufbau von technischen Gegenständen wie natürlichen Körpern scheinen aus dieser Perspektive in

---

einer neuen logischen Theorie – später Automatentheorie genannt – nötig ist, die sich eng an die Neurologie anlehnen soll (S. 311): „[...] a new, essentially logical, theory is called for in order to understand high-complication automata and, in particular, the central nervous system. It may be, however, that in this process logic will have to undergo a pseudomorphosis to neurology to a much greater extent than the reverse."

gleicher Weise beschreibbare mathematisch-logische Regelhaftigkeiten zu unterliegen. Das erlaubt, Organismen und Maschinen in bezug auf ihren körperlichen Aufbau zu vergleichen.

Exkurs: Von der behavioristischen Untersuchung zum axiomatischen Vorgehen

Die Sicht auf Organismen, die von Neumann hier vorschlägt, erinnert an die „behaviouristic study of natural events", die Rosenbluth, Wiener und Bigelow – mit denen von Neumann bereits länger in regem Gedankenaustausch stand – 1943 in „Behaviour, Purpose and Teleology" vorgeschlugen.[80] Dieses Papier gilt heute, zusammen mit dem besprochenen Artikel von McCulloch und Pitts (vgl. Abschnitt 2.2.2), als das erste Papier der Kybernetik. Ein kurzer Vergleich der behavioristischen Untersuchungsmethode von Rosenbluth, Wiener und Bigelow mit von Neumanns axiomatischem Verfahren macht deutlich, daß von Neumann sich behavioristische Sichtweisen zu eigen machte und sie so modifizierte, daß sie dazu dienen konnten, das Innere der Körper zu erforschen. Um zunächst ein Bild behavioristischer Ideen und ihres Stellenwertes in den USA der 40er Jahre zu geben, beginne ich mit einer kurzen Einführung zum Thema Behaviorismus. Für eine ausführlichere Behandlung der Entstehung und Inhalte des Behaviorismus verweise ich auf (Bruder 1982) und die sehr umfassende Besprechung von Sonntag (1988). Anschließend gehe ich auf „Behaviour, Purpose and Teleology" ein und reflektiere die inhaltlichen Beziehungen zu von Neumanns Arbeiten.

Behaviouristische Auffassungen waren in den USA vor dem Krieg weit verbreitet. Deshalb ist anzunehmen, daß alle Autoren mit grundlegenden behavioristischen Ideen in Berührung gekommen sind – allerdings weniger mit den Details der wissenschaftlichen Forschung, als vielmehr mit der populären Rezeption. Nachdem nämlich der heute zum Begründer des Behaviourismus stilisierte Watson die Universität verlassen

---

80 Die Theorie der Automaten weist große Ähnlichkeit mit Wieners Ideen zur Kybernetik auf (Burks 1987d, S. 364): „Von Neumanns theory of automata was closely related to Norbert Wiener's cybernetics;" 1948, dem Jahr des von Neumannschen Vortrags zur Automatentheorie, erschien Wieners Buch „Kybernetik". Im Kapitel „Rechenmaschinen und das Nervensystem" entwickelt auch Wiener nach dem Vorbild des Gehirns Ideen für den Bau von Rechenmaschinen. Zudem bespricht er Neuronen als Elemente, die wie Relais wirkten, wobei ihre Wirkung nach ihm auf dem „Alles-oder-Nichts"-Prinzip beruht (vgl. S. 152). Das Gehirn ist für ihn eine Rechen- und logische Maschine, eine Erkenntnis, die Wiener – wie McCulloch – wesentlich (Turing 1936) zurechnet (vgl. Wiener 1948, S. 157). Der Zusammenhang zwischen Wieners und von Neumanns Arbeiten wird ausführlich in (Heims 1987) erörtert. Ich gehe auf Wieners Arbeiten nur insoweit ein, als sie erlauben, den Orientierungsrahmen, der von Neumanns hier besprochenen Texten unterliegt, im Vergleich besser herauszuarbeiten.

mußte, weil er sich in eine Studentin verliebt hatte, schrieb er als Vizepräsident einer Werbeagentur zahlreiche populäre Artikel (Bruder 1982, S. 169):

> „Watson suchte diese [breite] Öffentlichkeit, nachdem ihm die wissenschaftliche versperrt war. Aber abgeschnitten von der Forschung, der Universität, dem Labor, mußte das, was ursprünglich Programm gewesen war, zur ‚Theorie' erklärt werden, tatsächlich zum Standpunkt des ‚Behavioristen' in allen möglichen anderen Fragen als denen der psychologischen Forschung. Und des ‚behaviourist's view' wurde zur fertigen Ansicht jedes beliebigen Problems: des ‚Gedächtnisses' und des ‚Denkens', der ‚Instinkte' und des ‚Unbewußten', der ‚Angst der Kinder' und der ‚Schwäche der Frauen', ‚warum Männer in 50 Jahren nicht mehr heiraten werden' und ‚was nach der Familie kommt' usw."

Zudem verwandte Watson behavioristische Reiz-Reaktionsmuster auch für die Erklärung gesellschaftlicher Prozesse. So schrieb er 1930 (Watson nach Sonntag 1988, S. 68): „Der Behaviourist glaubt, daß seine Wissenschaft eine Grundlage für die Ordnung und Kontrolle der Gesellschaft ist [...]." Als der öffentliche Vertreter des Behaviourismus beschränkte Watson sich also in seinen Aussagen nicht auf das menschliche Verhalten – ein Vorgehen, das sich allerdings nicht nur auf Watsons popularistischen Stil zurückführen läßt, sondern schon in der Form der behavioristischen Wissensaquisition angelegt war: Einen großen Teil ihrer Erkenntnisse gewannen behavioristische Wissenschaftler aus Experimenten mit Tieren (Ratten, Hunde etc.), deren Ergebnisse sie auf den Menschen übertrugen. Der Behaviourismus stand damit für eine „ ‚Psychologie der Organismen', worin Menschen und Tiere [und ganze Gesellschaften] zusammenfallen insofern, als sie sich ‚verhalten', und zwar nach den gleichen Gesetzen." (Sonntag 1988, S. 58) Er verwischte die Unterschiede zwischen Mensch und Tier und machte „Verhalten" zur zentralen Kategorie der Psychologie. Bewußtseinsprozesse wurden als unwesentlich oder nicht existent erklärt; statt dessen sollten „Reflexe" – in Muskeln und Nervensystem verankert – bestimmen, wie ein Organismus einen Reiz in eine Reaktion verwandelt (vgl. Sonntag, S. 65f.). Ein Reflex ist nach dieser Vorstellung zunächst angeboren und kann durch Konditionierung im Laufe der Zeit verändert werden. Die Frage danach, wie es dazu kommt, daß auf einen Reiz gerade diese Reaktion erfolgt, mußte vom behavioristischen Standpunkt aus durch Konditionierung erklärt werden bzw. erscheint als Frage danach, warum der entsprechende Reflex angeboren ist – und damit als Gegenstand biologischer Theorien wie der Evolutionstheorie. Die Frage nach dem Sinn des Handelns (die für meine Untersuchung der Entwicklung der Programmierung zentral ist) kann in der behavioristischen Theorie nicht beantwortet werden (Bohnsack 1995, S. 38):

> „Während Handeln im soziologischen Sinne sich – seit der Definition von Max Weber (1922) – durch seine subjektive Sinnhaftigkeit, seine Motiv- und Erwartungsgeleitetheit auszeichnet, ist für die Verhaltenstheorie die Frage nach dem Sinn des Handelns methodisch nicht lösbar, da dieser nicht exakt beobachtbar sei. Interaktion wird auf physisch beobachtbare ‚stimulus-response'-(Reiz-Reaktions-)Abläufe reduziert. Dabei hat der ‚Behaviorismus' das Ziel, Regelmäßigkeiten des Verhaltens von Organismen (also nicht nur Menschen) im allgemeinen zu erfassen."

Handeln ist also aus behavioristischer Sicht durch Reiz-Reaktions-Reflexe erklärbar. Vor dem Hintergrund dieses Handlungsbegriff erschienen zunächst menschliches und tierisches Verhalten als gleichartig. In einem weiteren Schritt ermöglichte dieser Handlungsbegriff – wie es in dem Artikel „Behavior, Purpose and Teleology", auf den ich nun eingehen werde, deutlich wird – technisches Operieren und menschliches Handeln in gleicher Weise zu erklären.

Das Papier „Behaviour, Purpose and Teleology" erschien 1943 in der „Philosophy of Science". Bereits 1942 hatte Rosenbluth es auf einem von der „Josiah Macy Foundation" gesponsorten Treffen in New York vorgestellt. Einer der Zuhörer bei diesem Vortrag war McCulloch, der zusammen mit Pitts bereits mit seiner Arbeit zur mathematischen Beschreibung des Nervensystems begonnen hatte und seine Ideen mit Rosenbluth, den er schon länger kannte, im Mai 1942 in New York diskutiert hatte.

Der Artikel kann als Auswertung der Arbeit gelesen werden, die Wiener und Bigelow im Rahmen der Entwicklung von Flugabwehrtechniken im Zweiten Weltkrieg leisteten und die ich deshalb – bevor ich auf den Artikel selbst eingehe – kurz beschreibe: Da Bomber die effektivste deutsche Waffe bei Angriffen auf europäische Städte und Einrichtungen waren und die damaligen Abwehrgeschütze relativ machtlos gegen diese kleinen und schnellen Ziele waren, wurde zu Beginn des Kriegs ein „Radiation Laboratory" am MIT eingerichtet, um die Flugabwehrtechniken zu verbessern. Wiener, damals bereits ein bekannter Mathematiker, arbeitete dort zusammen mit dem jungen Ingenieur Julian Bigelow, der selbst auch aktiver Pilot war, an dem „Anti-Aircraft-Predictor". Auf Grundlage einer Charakterisierung der Handlungen eines feindlichen Piloten sollte der AA-Prädiktor den weiteren Flug des von dem Piloten gesteuerten Flugzeuges im voraus berechnen und eine Rakete starten, um das Flugzeug abzuschießen (vgl. Galison 1997, S. 282). Das Flugzeug und der Pilot erschienen in ihren Überlegungen als organisch-technische Einheit (Galison 1997, S. 307):

> „Das Mensch-Flugzeug-Radar-Prädiktor-Artillerie-System war ein geschlossenes System, in dem es möglich erschien, Menschen durch Maschinen und Maschinen durch Menschen zu ersetzen. Für einen Flugabwehroperator handelte der Feind tatsächlich wie ein autokorrelierter Servomechanismus."

Doch die Überlegungen Bigelows und Wieners blieben nicht auf den Bereich der Flugabwehr beschränkt. Wiener betrachtete, so Galison (S. 296), „den AA-Prädiktor, bevor er überhaupt im Stande war, ein Flugzeug vom Himmel zu holen, als Prototypen für eine behavioristische Auffassung des Nervensystems."

„Behavior, Purpose and Teleology" ist das erste Papier, in dem Wiener und Bigelow in Zusammenarbeit mit dem Physiologen und Freund Wieners, Arturo Rosenbluth, die im Rahmen der Entwicklung des AA-Prädiktors angestellten Überlegungen verallgemeinern und auf Organismen und natürliche Ereignisse insgesamt

beziehen. Die in dem Artikel angestellten Überlegungen sind strukturiert von der Grundproblematik, an der Wiener und Bigelow zu dieser Zeit arbeiteten: der Frage danach, wie man eine Waffe so steuern kann, daß sie ein feindliches Mensch-Maschine-System (Flugzeug und Pilot) abschießen kann.

Definiert werden soll in dem Artikel die behavioristische Untersuchung natürlicher Ereignisse. Zudem wollen die Autoren die Wichtigkeit des Zweckbegriffs unterstreichen (Rosenbluth et al. 1943, S. 18):

> „This essay has two goals. The first is to define the behavioristic study of natural events and to classify behavior. The second is to stress the importance of the concept of purpose."

Betrachtet werden dabei nur äußere Relationen, wie das in einem Zusammenhang, in dem es um das Abschießen eines Ziels geht, naheliegt:

- Die behavioristische Untersuchung „consists in the examination of the output of the object and of the relations of this output to the input" (S. 18). Betrachtet wird also von außen beobachtbares Verhalten;
- Verhalten ist dem Text zufolge „any change of an entity with respect to its surroundings" (S. 18), ist also durch das Verhältnis zur Außenwelt charakterisiert;
- Auch „Zweck" bezieht sich ausschließlich auf das Erreichen eines vorgegebenen Ziels, das als zeitliche oder räumliche, von außen beobachtbare Beziehung zwischen zwei Objekten gefaßt wird (S. 18):

  > „The term purposeful is meant to denote that the act or behaviour may be interpreted as directed to the attainment of a goal – i. e. to a final condition in which the behaving object reaches a definite correlation in time or in space with respect to another object or event. Purposeless behavior then is that which is not interpreted as directed to a goal."

Möglich wird auf Grundlage dieser Begrifflichkeit, Dingen, die sich in bezug auf ihre Umgebung verändern, zunächst Verhalten, aber auch das Anstreben eines Ziels und damit zweckhaftes Handeln zuzuschreiben. Ein Stein beispielsweise, der von einem Dach fällt, „verhält" sich, da er sich in bezug auf seine Außenwelt verändert; und sein Verhalten kann als zielstrebig oder zweckhaft interpretiert werden, weil er nach einer gewissen Zeitspanne ein bestimmtes Verhältnis zu z. B. einem auf dem Bürgersteig entlanggehenden Menschen erreicht – ihm etwa vor die Füße fällt.[81]

Die Fokussierung auf das Ein-/Ausgabeverhalten läßt in dem Artikel Differenzen zwischen dem Verhalten von Organismen und Maschinen in den Hintergrund treten. Nach einer nur vierseitigen Analyse glauben die Autoren, ähnlich wie von Neumann für seine „axiomatic procedure", „that a uniform behavioristic analysis is applicable to both machines and living organisms, regardless of the complexity of behaviour." Denn (S. 22):

---

81 Galison gibt eine längere Diskussion wider, die Wiener mit einem jungen Philosophen, Taylor, zu diesem Zweckbegriff führte (Galison 1997, S. 303ff.): „Für Taylor öffnete die Willkür der Formel ‚kann so interpretiert werden' Tür und Tor für ein Verständnis der fallenden Ziegelsteine und der stehengebliebenen Uhren als zweckgerichtet." Wiener, Bigelow und Rosenbluth hielten jedoch, trotzdem ihre Argumentation recht brüchig wirkt, an ihrem Zweckbegriff fest.

„Such qualitative differences [between characteristics of living organisms and machines] have not appeared so far. The broad classes of behaviour are the same in machines and in living organisms."

Auch über die Konstruktion von Organismen durch Menschen spekulieren sie und orakeln für die Zukunft (S. 23):

„If an engineer were to design a robot [today ... he] would probably build it out of metallic parts, some dielectrics and many vacuum tubes [...]. In future years, as the knowledge of colloids and proteins increases, future engineers may attempt the design of robots not only with a behaviour, but also with a structure similiar to that of a mammal. The ultimate model of a cat is of course another cat, whether it be born of still another cat or synthesized in a laboratory."

Wie von Neumann haben also auch Rosenbluth, Wiener und Bigelow im Sinn, daß man lebende Organismen künstlich herstellen oder nachbauen könnte. Allerdings haben Säugetiere auf Grund ihrer andersgearteten Materialität bei ihnen auch eine andere Struktur und sie unterscheiden „behaviouristic approach" und „functional analysis" (S. 18):

„The behaviouristic approach consists in the examination of the output of the object and of the relations of this output to the input. [...] In a functional analysis, as opposed to a behaviouristic approach, the main goal is the intrinsic organization of the entity studied, its structure and its properties".

Im Unterschied zu von Neumann sind die „black boxes" von Rosenbluth, Wiener und Bigelow die ganzen Organismen und Maschinen, also die vollständigen, sichtbaren Entitäten. Als die wesentliche Ebene, auf der sich Maschinen und Organismen gleichen, konstruieren sie das nach außen sichtbare Verhalten. Von Neumann hingegen verlagert den Vergleich in das Innere der Organismen bzw. Maschinen hinein. Die behavioristischen Verfahren dringen bei ihm gewissermaßen in den Körper ein. Seine „black boxes" sind die konstitutiven Elemente, aus denen sich Automaten zusammensetzen. Seine Betrachtung umfaßt also auch die innere Organisation und damit Teile der „functional analysis", die nach Rosenbluth, Wiener und Bigelow noch beträchtliche Unterschiede zwischen lebenden Organismen und Maschinen zeigt (Rosenbluth et al. 1943, S. 22): „While the behaviouristic analysis of machines and living organisms is largely uniform, their functional study reveals deep differences." Zwar sind auch nach von Neumann die konstitutiven Elemente für Organismen und Maschinen in bezug auf ihre Materialität verschieden, vor dem Hintergrund der „axiomatic procedure" scheint die Art ihres Funktionierens als Stimulus-Response-Automatismen sich jedoch zu gleichen. Nicht nur in Hinblick auf von außen beobachtbares Verhalten, sondern auch in bezug auf ihre interne Organisation scheinen sich Organismen und Maschinen damit in wesentlichen Aspekten zu ähneln. Die interne Organisation kann so bei von Neumann als Ebene gesehen werden, auf der Organismen und Maschinen verglichen werden können.

Wie der Exkurs verdeutlicht, unterliegen der „General and Logical Theory of Automata“ behavioristische Sichtweisen. Jedoch wird nicht nur, wie das in behavioristischen Untersuchungen und auch in „Behavior, Purpose and Teleology“ in der Regel der Fall war, von außen beobachtbares Verhalten untersucht. Organismen werden konzipiert als bestehend aus sehr vielen, klar voneinander abgrenzbaren Einzelelementen. Von diesen wird angenommen, daß sie als Stimulus-Response-Automatismen beschreibbar sind. Auf Grundlage der „axiomatic procedure“ erscheint ein Organismus als Struktur, die sich dadurch konstituiert, daß viele Elemente in ihrem reflexartigen Ein-/Ausgabeverhalten zusammenwirken. Aus einer behavioristischen Sicht, nach der Verhalten durch das Zusammenwirken von angeborenen oder konditionierten Reflexen entsteht, läßt sich das Verhalten und Handeln des Organismus so erklären. Das nach außen beobachtbare Verhalten des menschlichen Nervensystems bzw. des Menschen – nach von Neumann vollständig und eindeutig beschreibbar – wird so gesehen durch das Zusammenspiel kleinerer – ebenfalls vollständig und eindeutig beschreibbarer – Verhaltenskomplexe, die den Reflexen vergleichbar sind, hervorgebracht.

In Zusammenhang mit dieser Sicht, in der Organismen als Aggregate sehr vieler, in ihrem Verhalten eindeutig und vollständig beschreibbarer Elemente erscheinen, bezeichnet von Neumann die Objekte selbst auch als ihre eigene Beschreibung. Für Verhaltensformen hoher Komplexität wie z. B. dem Analogieerkennen hält er fest, daß das „real object“ – in diesem Falle das „visual brain“ – durchaus schon die einfachste Beschreibung seiner selbst sein könnte (von Neumann 1948, S. 311):

> „We are dealing here with parts of logics with which we have practically no past experience. The order of complexity is out of all proportion to anything we have ever known. We have no right to assume that the logical notations and procedures used in the past are suited to this part of the subject. It is not at all certain that in this domain a real object might not constitute the simplest description of itself;“

Im Bereich hoher Komplexität ist also möglicherweise das Objekt selbst seine einfachste Beschreibung, was insbesondere bedeutet, daß das Objekt selbst auch als Beschreibung gedacht wird. Die Differenz Objekt-Beschreibung verschwimmt. Das dokumentiert sich auch darin, daß einige Zeilen später beispielhaft als „real object“ nicht, wie eigentlich zu erwarten, das „visual brain“ benannt wird, sondern dessen „connection pattern“. Das Verbindungsmuster der Neuronen, also bereits eine Abstraktion von dem Materiellen, steht hier für das reale Objekt, das „visual brain“.

In dem Papier werden nicht nur Organismen und Maschinen als aufgebaut aus vielen, in ihrem Verhalten vollständig beschreibbaren Einzelelementen gesehen. Ähnlich wird auch Verhalten – und damit zeitliche Prozesse – als unterteilbar in einzelne Phasen oder Schritte betrachtet, die als vollständig und eindeutig beschreibbar angesehen werden (S. 310):

> „There is no doubt that any special phase of any conceivable form of behaviour can be described ‚completely and unambiguously‘ in words. This description may be lengthy, but it

is always possible. To deny it would amount to adhering to a form of logical mysticism which is surely far from most of us."

Von Neumann charakterisiert, wie oben aufgezeigt, seine Untersuchung auch als eine, die aus dem Blickwinkel der Rechenmaschine vorgenommen wird. Er wendet eine atomistische Perspektive nicht nur auf Organismen, sondern auch auf zeitliche Prozesse an. Das macht deutlich, daß aus der Perspektive der Rechenmaschine gesehen die Organismen die Struktur der Rechenmaschine, so wie sie von Neumann besonders 1945 beschreiben hatte, widerspiegeln: sie erscheinen als aufgebaut aus als „Stimulus-Response-Automatismen" beschreibbaren Elementen. Ebenso scheinen zeitliche Prozesse in diskrete Abschnitte zerlegbar zu sein. Aus der Sicht einer Maschine, die selbst in einem bestimmten Zeittakt arbeitet, der die Operationen ihrer als „Stimulus-Response-Automatismen" beschreibbaren Bauteile synchronisiert, ist das verständlich.

Während das Gerät des „First Draft" als nervensystemähnlicher Organismus gesehen wird, erscheinen in „The General and Logical Theory of Automata" Organismen und besonders das Nervensystem als eine Art technische Mechanismen. Mit der Annahme der vollständigen und eindeutigen Beschreibbarkeit des Verhaltens der Neuronen geht einher, daß das Nervensystem bzw. hier ein Teil davon – das „visual brain" – auch als Struktur, als Verbindungsmuster bzw. als Beschreibung gedacht wird. Wie im „First Draft" dokumentiert sich in dem Oszillieren der Perspektiven die Hybridobjekt-Sichtweise: Das menschliche Nervensystem und Gehirn als natürlicher Organismus wird technisch – als Mechanismus – verstanden und kann auf dieser Grundlage als Struktur und Beschreibung – als Symbolsystem – konzipiert werden.

Voraussetzung der „axiomatic procedure" ist die Annahme, daß das Verhalten der einzelnen Elemente tatsächlich vollständig und eindeutig beschreibbar ist. Auf Grundlage einer technischen Sicht auf die Natur, in der Neuronen wie Vakuumröhren als „basic switching organs" erscheinen, ist diese Voraussetzung erfüllt. Auch das Nervensystem erscheint als Äquivalent eines Mechanismus, und das von ihm hervorgebrachte Verhalten zumindest prinzipiell als vollständig und eindeutig beschreibbar. Diese Sicht auf das menschliche Denken stand bei von Neumann in einem breiteren Rahmen. Schon 1931 hielt er fest, daß (von Neumann nach Mehrtens 1990, S. 295)

„die philosophisch-erkenntnistheoretische Frage im Begriffe ist, sich in eine mathematisch-logische zu verwandeln. [... Die] Vorstöße im Gebiet der mathematischen Logik haben es mit sich gebracht, daß heute in den Grundlagenfragen immer mehr eindeutige mathematische Fragestellungen und nicht Geschmacksunterschiede zu untersuchen sind."

Auch philosophische Fragen sind demnach – und mit Fortschreiten der Logik mehr und mehr – in mathematisch-logische Fragen umwandelbar, die mathematisch präzise beantwortbar scheinen. Philosophisch-erkenntnistheoretisches Denken scheint sich so gesehen weitgehend in mathematischer Logik zu erschöpfen, wie dies ja auch bei McCulloch und Pitts deutlich wurde. Rechenmaschinen konnten zwar nur rechnen, in

einer Welt, so wie sie in von Neumanns Texten konzipiert ist, deckt dieses Rechnen das Denken jedoch weitgehend ab. Die Rechenmaschine erscheint so als „Prototyp" beim Bau von Automaten, die so flexibel sind wie natürliche Organismen und besonders das menschliche Nervensystem (s. o).

Es hatte den Anschein, als ob von Neumanns Maschinen mehr waren als nur Rechenmaschinen, ihre Art zu funktionieren schien auf den biologischen Bereich übertragbar. Organismen konnten anscheinend als Maschinen beschrieben werden. Die Technik erscheint tendenziell als eine Art Theorie. Die immateriellen Maschinen, die so denkbar wurden, behielten bis heute Charakteristika aus beiden Bereichen – dem materiellen sowie auch dem aus Zeichen bestehenden. So kann man heute in dem Duden der Informatik unter „Automat" nachlesen:

> „**Automat**: Im allgemeinen versteht man unter einem Automaten ein technisches oder mechanisches Gerät, das zu einer Eingabe ein bestimmtes Ergebnis ausgibt. Beispiele: Kaugummiautomaten [...]. In der Informatik bezeichnet man als Automaten vorwiegend mathematische Modelle von Geräten, die Informationen verarbeiten und dabei Antworten auf Eingaben geben. Der Zweig der *theoretischen Informatik*, der sich damit beschäftigt, wird *Automatentheorie* genannt."

Doch sind die Automaten genannten mathematischen Modelle, obwohl ja Modell und damit Zeichen, auch zu Aktionen und Tätigkeiten, zu Bewegung fähig. Einige Zeilen weiter heißt es:

> „Ein **erkennender Automat** (sog. *Akzeptor*) für eine Sprache L ist ein Automat, der ein Wort w einliest und anschließend anzeigt, ob w in L liegt. Ein übersetzender Automat (sog. *Transduktor*) berechnet zu einem gegebenen Wort w ein Ausgabewort v. Man kann sich einen erkennenden Automaten als ein Gerät vorstellen, das ein Eingabewort auf einem Eingabeband einliest, eine Berechnung durchführt und dann meldet, ob das Eingabewort in der vorgegebenen Sprache L liegt."

Automaten sind so gesehen immaterielle Maschinen, Hybride aus widerständigem, tätigem (lebendigem) Materiellem und Zeichen. Diese immateriellen Maschinen waren vielleicht von Neumanns beständigstes Produkt: Sie blieben der Informatik erhalten, auch wenn der EDVAC oder die IAS-Maschine mittlerweile nur noch im Museum zu besichtigen sind.

3.1.2 Materielle Beschreibungen

Während einerseits materielle Körper und ihre Beschreibung in „The General and Logical Theory of Automata" nah aneinander gedacht werden – Körper in diesem Sinne immaterialisiert werden –, erhalten in einer Art Gegenbewegung Beschreibungen materiellen Charakter. Für diesen Umstand ist kennzeichnend, daß Beschreibungen mit Instruktionen und, in einem weiteren Schritt, Instruktionen und Gene verglichen und als ähnlich befunden werden, worauf ich in diesem Abschnitt eingehe.

Am Ende der „General and Logical Theory of Automata“ stellt von Neumann die Frage, wie ein Automat einen anderen produzieren kann, der genauso komplex oder gar komplexer ist als er selbst. Es geht um die „self-reproduction of automata“ (vgl. von Neumann 1948, S. 315) und eine Imitation des natürlichen Evolutionsprozesses, in dem sich Organismen zu immer höherer Komplexität entwickelten (vgl. S. 312).[82] Dabei lehnt sich von Neumann explizit an Turings Papier von 1936 an und gibt Turings Idee von der universellen Maschine, modifiziert durch seine Automatenterminologie, relativ ausführlich wieder (vgl. S. 313f.). Der sich selbst reproduzierende Automat, den von Neumann darauf folgend entwirft, tut auf den ersten Blick nicht viel: Er erhält eine „Beschreibung“ von sich selbst, dupliziert diese, stellt sich selbst auf Grundlage der Beschreibung her, führt die verdoppelte Beschreibung in den neu entstandenen Automaten ein und löst dieses Konstrukt (das Duplikat seiner selbst versehen mit seiner eigenen Beschreibung) von sich ab, so daß der ganze Vorgang wieder von vorne beginnen kann.

Interessant ist, wie von Neumann dieses Geschehen darstellt. Die Beschreibung eines Automaten wird auch „instruction“ (abgekürzt „I“) genannt; indem Beschreibung und Instruktion gleichsetzt werden, werden Deskription und Aufruf zur Operation gemischt. Die Instruktionen-Beschreibungen bestehen, ähnlich wie die Organismen und Maschinen, aus einer Kette von „structural elements“, auch „elementary parts“ oder „machine parts“ genannt, wobei ca. 12 verschiedene solcher Elementarteile ausreichten „to permit the construction of the wide variety of mechanisms here required“ (S. 315). Nach dieser Vorstellung sind Beschreibungen bzw. Instruktionen und Automaten also aus den gleichen materiellen Bauteilen zusammengesetzt. Die Selbstreproduktion des Automaten findet nun wie folgt statt (S. 316):

> „The constructing automaton is supposed to be placed in a reservoir in which all elementary components in large numbers are floating, and it will effect its construction in that milieu. [... The] instructions, as pointed out, will have to be aggregates of elementary parts.“

In einem flüssigen Milieu, in dem in großer Zahl elementare Teile vorhanden sind, treibt also die Instruktion als Kette von Elementarteilen in einen Automaten, der für diesen Zweck einen speziellen Platz besitzt (ebd.): „All of them will have a place for an instruction I, that is, a place where such an instruction can be inserted.“ Der Automat beginnt dann, wie oben beschrieben, sich immer wieder selbst zu reproduzieren. Daß auf diese Weise nicht nur Selbstreproduktion passieren kann, sondern auch komplexere Automaten entstehen können als der Automat selbst, war nach von Neumann kein Problem (ebd.):

> „One need not to worry about how a fixed automaton of this sort can produce others which are larger and even more complex than itself. In this case the higher complexity of the object to

---

82 Diese Idee des Nachvollzugs der Schöpfung spielte auch später in der Programmierung und besonders in der Künstlichen Intelligenz eine große Rolle (vgl. Stach 1995).

be constructed will be reflected in the presumably still greater size of the instructions I that have to be furnished."

Die dargestellte Vorstellung von der Automatenreproduktion ist also recht plastisch und vermag durchaus, an den natürlichen Reproduktionsvorgang zu erinnern, bei dem nach klassischer Vorstellung das Weibliche (hier der aufnehmende Automat) als Gefäß den männlichen Samen (hier die möglicherweise komplexeren Instruktionen) nur ausbildet.[83]

Von Neumann selbst vergleicht am Ende des Textes die „instructions" zwar nicht mit dem männlichen Samen, dafür aber mit einer anderen Instanz, die in der Moderne den Ursprung des Lebens in sich zu tragen scheint, mit dem biologischen Gen (S. 317): „For instance, it is quite clear that the instruction I_D is roughly effecting the functions of a gene." Als fundamentaler Akt der Reproduktion seiner Automaten gilt entsprechend nicht etwa die Herstellung des neuen Automaten, sondern die Duplikation seiner Beschreibung, der „instruction" (ebd.):

„It is also clear that the copying mechanism B performs the fundamental act of reproduction, the duplication of the genetic material, which is clearly the fundamental operation in the multiplication of living cells."

Die Fähigkeit von Genen, sich selbst zu reproduzieren, galt 1948 als eine der wenigen Gewißheiten in bezug auf die Frage, wie Gene die Entstehung des Lebens steuern. Im American Scientist schrieb z. B. Sturtevant, ein bekannter amerikanischer Genetiker, im April 1948 (S. 236): „One thing that is definitely known about genes is that they reproduce themselves. At least once per cell division, on the average, each gene somehow conditions the formation of a copy of itself." Obwohl man wenig darüber wußte, wie Gene dies taten – woraus z. B. Gene bestanden, welches also ihre materielle Basis war, wurde erst 1953 von Watson und Crick erklärt –, galten sie als die biologische Einheit, die die Entwicklung und Entstehung des Lebens steuerte. So konstatierte Sturtevant 1948 einleitend (S. 225): „Every individual organism is the resultant of the activity of the ultimate hereditary units, or genes, that it carries and of the environmental conditions under which it has developed."

In „Refiguring Life" beschreibt Keller, wie in den USA der 30er und 40er Jahre die Genetik das Bild vom Gen als alleinigem Träger der Kontrolle über die Entwicklung eines Organismus durchsetzte – trotz guter Gegenargumente gegen diese Vorstellung. Die allgemeine Unklarheit darüber, wie ein Gen aufgebaut war und wie Gene eigentlich die Entwicklung des Lebens steuerten, wurde nach Keller durch den Begriff der „gene action" kompensiert; der Diskurs über „gene action" ermöglichte, daß die Wissenschaftler mit ihrer Arbeit fortfuhren, ohne daß ihre Wissenslücken als Hindernis wahrgenommen werden mußten. Ähnlich wie die Gene die Entwicklung des Lebens vermögen in von Neumanns Text zur Automatentheorie die Instruktionen allein die

---

83 Phantasien aus Raumfahrt, Militär und Wissenschaft, die sich um die Selbstreproduktion von Automaten und Robotern ranken, schildert (Coy 1985, S. 119ff.).

Selbstreproduktion des Automaten zu steuern. Sie werden als handelnde Entitäten gezeichnet, ohne daß dies als Problem wahrgenommen wird. Sie sind – ähnlich wie das Gen, so wie es damals vorgestellt wurde, – janusgesichtig und vereinen materielle und geistige Charakteristika (Keller 1995, S. 9f.):

> „[...] it is part physicist's atom and part Platonic soul – at one and the same time a fundamental building block and an animating force. Only the action of genes can initiate the complex manifold of processes comprising a living organism."

Die „Instruktionen" in von Neumanns Papier zur Automatentheorie – als „Beschreibungen" Plan von Automaten, als Instruktionen zur Operation auffordernd und als bestehend aus „elementaren" oder auch „Maschinenteilen" – zeigen die gleiche Janusgesichtigkeit, die Keller für die damalige Vorstellung vom Gen beschreibt. Sie sind fundamentale Bausteine der neu entstehenden Automaten und besitzen die Fähigkeit, zu agieren, den Reproduktionsvorgang in Gang zu setzen und zu steuern. Die Instruktionen sind Beschreibungen, aber auch Gegenstände, die agieren. In ihnen fällt Objekt und Beschreibung, aber auch animierende Kraft zusammen. Ähnlich wie in der Genetik der Zeit – und ähnlich wie in behavioristischen Modellen die Frage nach dem Sinn des Handelns nicht beantwortet wird – werden offene Fragen danach, woher die Triebkraft stammt, die Instruktionen veranlaßt, einen neuen Automaten zu kreieren, und wie Beschreibung, Operation sowie materielle und geistige Charakteristika in nur einer Einheit aufgehoben sind, unbeachtet gelassen.

3.2 Automaten und Programme

Bis heute gilt die Konstruktion von Automaten als ähnliche Tätigkeit wie das Schreiben eines Programms. Bereits 1958 charakterisiert McCarthy die Konstruktion eines Automaten und die Programmierung in einer Programmiersprache als gleichwertige Möglichkeit zur Repräsentation von Verhalten (McCarthy 1958, S. 404):

> „All behaviors must be representable in the system. Therefore, the system should either be able to construct arbitrary automata or to program in some general-purpose programming language."

Auch in einem Lehrbuch zur Informatik von 1992 erscheinen das Programmieren und die Konstruktion von Automaten als ähnliche Tätigkeiten. Es gibt, so heißt es hier, Automaten, die Teil von Programmen sind, oder auch andere, die sehr leicht in Programme umgesetzt werden können (Pepper 1992, S. 128, S. 131):

> „Auch in ganz ‚normaler' Software können endliche Automaten oft sehr gut zur Verarbeitung der Benutzereingabe eingesetzt werden, insbesondere bei Plausibilitätskontrollen. [...] Die programmtechnische Umsetzung eines solchen Automaten ist übrigens recht einfach."

Da die Konstruktion von Automaten und die Programmierung bis heute als verwandte Tätigkeiten begriffen werden, kann die Art und Weise, wie von Neumann Automaten konzipierte, auch etwas über die Tätigkeit des Programmierens aussagen – zumal von

Neumann selbst sein Papier auch als Analyse des Problems der Organismen und Organisation aus der Perspektive der Rechenmaschine charakterisierte (von Neumann 1948, S. 290): „It is nevertheless of a certain interest to analyse the problem of organisms and organization from the point of view of these relatively small, artificial automata". Ich werde in diesem Abschnitt diese Sichtweise in Hinblick darauf auswerten, was sie zu einer Charakterisierung von Programmen und der Tätigkeit des Programmierens beitragen kann.

Die in diesem Abschnitt angestellten Überlegungen kreisen um ein Thema, das bereits in den im letzten Kapitel besprochenen Texten deutlich wurde: Die Instruktionen des Codes, die zur Programmierung der Rechenmaschine dienen sollten, erscheinen in diesen Texten einerseits als technische Artefakte, andererseits als Symbole und Zeichenketten mit mathematischer Bedeutung, die z. B. die logische und algebraische Definition eines Problems beschreiben sollen. Ihnen wird die Fähigkeit zu operieren – insbesondere zu steuern – zugeschrieben. Sie sind damit den Instruktionen vergleichbar, die in „The General and Logical Theory of Automata" bei der Selbstreproduktion von Automaten eine zentrale Rolle spielen und dort mit Genen verglichen werden. Es handelt sich bei diesen Instruktionen um einen neuartigen Typ von Notationen, die auch gegenständlich gedacht werden und vor allem die Fähigkeit zur Aktion zu besitzen scheinen.

Zeichen und Schrift als Gegenstand der Operation wahrzunehmen, ist in der Mathematik, z. B. in mathematischen Kalkülen, gängig. Die Symbole stehen hier für die Möglichkeit, sie zu manipulieren wie andere Gegenstände auch, können aber gleichzeitig Träger einer zu interpretierenden Bedeutung sein (Krämer 1997, S. 118):

> „Ein kalkülisierter Zeichenausdruck kann in einer zweifachen Perspektive thematisiert werden. Als Symbolkonfiguration nimmt er eine gewisse Stelle in Raum und Zeit ein und ist schematisch ‚bearbeitbar' wie andere Dinge auch. Als interpretierbarer Ausdruck repräsentiert die Symbolkonfiguration einen Satz, der eine extrasymbolische Bedeutung hat wie andere Sätze auch."

Zeichen können also zum einen für etwas zweites stehen – repräsentieren – und in diesem Sinne eine Bedeutung habe; sie können aber auch als „operative Symbole" gesehen werden, „wenn mit ihnen nach Regeln verfahren wird, die nicht mehr Bezug nehmen auf das, was die Zeichen bedeuten, sondern nur ihre syntaktische Gestalt" (Krämer 1988, S. 72). Wenn ein Programm in einem Rechenmaschinenspeicher abgelegt ist und anschließend bearbeitet wird, kann es in dieser Weise interpretiert werden: es werden Zeichen im Speicher technisch repräsentiert und schließlich von der Maschine nach der Maschine technisch inkorporierten Regeln manipuliert. Die Bedeutung der Zeichen spielt für diesen Prozeß keine Rolle.

Doch wer ist in diesem maschinellen Prozeß die steuernde Instanz? Während operative Symbole manipuliert oder bearbeitet werden, erscheint der Code in den im letzten Kapitel besprochenen Texten als steuernde, selbst manipulierende Instanz.

Wenn auch an einigen Stellen zumeist implizit deutlich wurde, daß die zentrale Steuereinheit für die Kontrolle der Maschine ebenfalls eine Rolle spielt, wurde die steuernde Kraft letztendlich bei den Instruktionen verortet. Es handelt sich bei den Codewörtern um scheinbar selbst agierende Notationen.

In den folgenden beiden Abschnitten werde ich diesen Typ Notation, der in der Programmierung eine zentrale Rolle spielt, näher betrachten. Dazu gehe ich zunächst (3.2.1) noch einmal genauer auf den Zusammenhang Instruktion-Gen ein, so wie er in „The General and Logical Theory of Automata" formuliert ist. Vor allem betrachte ich die Übernahme der Kodier- und Programmiermetaphorik in die Genetik und Molekularbiologie. Anschließend reflektiere ich, wie diese Rezeption auf den Bereich der Programmierung und das Sinn- und Deutungsgeflecht, in das diese eingebettet ist, zurückwirkt: Ähnlich wie Lebewesen qua Programmierung zu entstehen scheinen, wirkt das Programmieren nun wie das Konstruieren handelnder und gegenständlich gedachter Entitäten. In Abschnitt 3.2.2 werte ich die „axiomatic procedure", die von Neumann 1948 vorstellte, in Hinblick auf die Programmierung aus. Deutlich wird, wie aus einer behavioristisch-atomistischen Sicht Programme als handelnde Instanzen denkbar werden.

3.2.1 Instruktionen und Gene

Die sich in den in Kapitel 2 besprochenen Texten zu Rechenmaschinen dokumentierende Vorstellung, daß Instruktionen die Maschine steuern, wird in „The General and Logical Theory of Automata" in einen breiteren Rahmen eingebettet. Instruktionen haben hier, wie für das Gen vorgestellt, gegenständlichen Charakter und die Kraft zur Operation. Als solche können sie, ähnlich wie für das Gen und Organismen vorgestellt, die Selbstreproduktion eines Automaten steuern. Eine Instruktionenfolge erscheint als autarke Entität, die Handlungsfähigkeit besitzt.

Instruktionen an die Rechenmaschine, der „Code", scheinen vor diesem Hintergrund im Rechner in etwa das zu vollziehen, was Gene im Organismus tun – auch wenn bzw. gerade weil in den 40er Jahren recht unklar war, was genau Gene eigentlich taten und wie sie gegenständlicher Baustein und Modell und damit Beschreibung und ausführende Kraft in einem sein konnten (vgl. Abschnitt 3.1.2). Die Kodier- und Programmiermetaphorik wurde entsprechend in den 50er Jahren in die Genetik importiert, was auch dadurch begünstigt wurde, daß von Neumann mit Molekularbiologen zusammenarbeitete (Kay 1996, S. 6):

> „Von Neumann made contacts with the biomedical community, participating in meetings and communicating with life scientists (e.g. Max Delbrück, Sol Spiegelmann, Joshua Lederberg and John Edsall). Through these exchanges he was encouraged to develop his self-duplicating machine as possible heuristic model of gene action. Although von Neumann's project in

biology did not go far (he died in 1957), its discursivities began to permeate genetic representations."

In dem Artikel, in dem Watson und Crick 1953 die Entdeckung des materiellen Aufbaus von Genen, der DNA, vorstellten, benutzten sie bereits die Code-Metaphorik (Watson, Crick 1953, S. 966):

„It follows that in a long molecule many different permutations are possible, and it therefore seems likely that the precise sequence of bases is the code which carries the genetic information."

Entsprechend stellt Keller (1995, S. 18) fest:

„Genetics and molecular biologists were euphoric: there, surely, must be the answer! DNA carries the ‚genetical information' (or program), and genes ‚produce their effects' by providing the ‚instructions' for protein synthesis."

Lily E. Kay arbeitet in ihrem Papier „How the Genetic Code Became an Information System" heraus, daß die Code- und Programmiermetaphorik aus den im Entstehen begriffenen Informations- und Computerwissenschaften und der Kybernetik importiert wurde (Kay 1996, S. 1):

„In the 1950s molecular biology underwent a striking discoursive shift: It began to represent itself as a communication science, allied to cybernetics, information theory, and computers. Through the introduction of such terms as information, feedback, messages, codes, alphabet, words, instructions, texts, and programs molecular biologists came to view organisms and molecules as information storage and retrieval systems. Heredity came to be conceptualized as electronic communication – akin to guidance and control systems – governed by a genetic code [...]. This semiotic and linguistic repertoir was absent from molecular biology before the 1950s. Based on these scriptual representations the genome could be read and edited unambiguously by those who know, laying claims to new levels of control over life: beyond control of matter there was now control of information and the power of the word."

Im Diskurs der Genetik erschien Vererbung also als bewirkt durch eine Art Computerprogramm, Leben – ähnlich wie der Rechenmaschinenbau und die Kodierung bei von Neumann – nicht zuletzt als eine Angelegenheit der Logik. So schrieb z. B. der Nobelpreisträger Francois Jacob ein Buch mit dem Titel „The Logic of Life" und stellt in der Einleitung fest (nach Kay 1996, S. 2): „[...] heredity functions like the memory of a computer." In „The Book of Life" hielt der Vorkämpfer für das Projekt der Entschlüsselung des menschlichen Genoms, Robert Sinsheimer 1967 fest (nach Kay 1996, S. 2):

„In this book [the book of life] are instructions, in a curious and wonderful code, for making a human being. In one sense – on a subconscious level – every human being is born knowing how to read this book in every cell of his body. [...] we have begun to understand the content of these books and the language in which they are written."

Der Körper erscheint in der Genetik und Molekularbiologie – ähnlich wie Organismen in „The General and Logical Theory of Automata" – als Hybridobjekt: er ist natürlich, wird jedoch als technisch und schließlich als Zeichensystem konzipiert (Keller 1995, S. 118):

„The body of modern biology [...] has become just another part of an informational network, now machine, now message, always ready for exchange, each for the other."

Den Organismus im wesentlichen als eine Art Buch oder Nachricht zu sehen, läßt die Trennung zwischen Objekt und Beschreibung verschwimmen. Ihn als Buch (umzu)-schreiben oder ihn als Körper (umzu)konstruieren scheint kaum noch einen Unterschied zu machen (Rheinberger 1997, S. 269):

> „Dazu überzugehen, lebende Systeme als informationsverarbeitende Strukturen zu sehen anstatt als mechanische Apparate oder als Energie wandelnde und dissipierende Maschinen, das unterscheidet die Biologie der zweiten Hälfte des 20. Jahrhunderts von den Organismusvorstellungen des 17./18. und des 19. Jahrhunderts. Den Organismus als ein Buch zu lesen ist aus einer bloßen Analogie zu einer wörtlich zu nehmenden Beschäftigung geworden. [...] Das Programm der Molekularbiologie [...] läuft darauf hinaus, ihn zu schreiben und umzuschreiben. Das wird uns in dem Augenblick bewußt, wo wir gewahr werden, daß die Molekularbiologie die Möglichkeit schafft, auch unseresgleichen auf der Ebene der Gene umzuschreiben."

Wird der Organismus als Buch oder – ähnlich wie bei von Neumann – ein Organismus auch als seine Beschreibung gesehen, stellt sich die Frage danach, was eine Beschreibung repräsentiert, neu (Rheinberger et al. 1997):

> „[...] das traditionelle Verhältnis von Repräsentation und Referenz [wird] geradezu umgestülpt [...], indem die molekulare Schrift selbst gar nicht mehr als Darstellung von *etwas* gedacht werden kann, sondern zu dem primordialen Vorgang wird, der Repräsentanten überhaupt erst erzeugt."

Was bedeuten diese Überlegungen zur Genetik nun für das Programmieren? Der Import der Kodier- und Programmiermetaphorik in die Genetik – die Vorstellung des Lebens als kodiert oder programmiert – wirkte auch darauf zurück, wie Programme und das Programmieren in der Folgezeit wahrgenommen wurden und mit welchem Hof von Konnotationen diese Begriffe umgeben waren und bis heute sind. Programme konnten in ein Sinngeflecht eingebettet werden, welches sie als Abbild und – mehr noch – als Nachvollzug, Rekonstruktion und zuweilen gar Verbesserung dessen erscheinen ließ, was in der Welt und besonders bei der Entstehung des Lebens passiert. Sehr prägnant formulierte z. B. Minsky (Minsky nach Coy 1985, S. 146):

> „Es macht mich krank, die Lobreden auf die Evolution zu hören; kein Programmierer, der sich selbst achtet, würde *Software*-Fehler solch schrecklicher Art stehen lassen! Ich wette, es wäre mindestens genauso gut, ganz neu anzufangen (ohne den Dreck, der sich in Milliarden Jahren angesammelt hat) und alles, was wir wirklich von diesem riesigen Zeichen-Verarbeitungs-Strukturnetz, das wir unser *Selbst* nennen, in einen stabileren und schöneren unsterblichen Code zu übertragen."

Das Selbst erscheint hier als kodiertes, zeichenverarbeitendes Strukturnetz – als eine Art Automat –, der Körper des Selbst tritt vollkommen in den Hintergrund. 1985 stellt Wolfgang Coy entsprechend einen Siegeszug der Ideologie der Information fest (Coy 1985, S. 142):

> „Die Ideologie der Information hat sich durchgesetzt – die Details der Materie verschwinden hinter der logischen Funktion. Die Verwechslung von Informationsabbild und Original wird nicht nur in der Datenverarbeitung zum allseits akzeptierten Irrtum."

Das Kodieren und Programmieren – als das Niederschreiben scheinbar agierender Instruktionen, die Ähnliches wie das Gen zu leisten scheinen – wird auch als

Schöpfung von Objekten denkbar, die körperlich-materiell konnotiert sind. In einer vielzitierten Textstelle hat Joseph Weizenbaum eine solche Wahrnehmung des Programmierens 1977 wie folgt zum Ausdruck gebracht (Weizenbaum 1977, S. 160):

> „Der Programmierer ist jedoch der Schöpfer von Universen, deren alleiniger Gesetzgeber er selbst ist. [...] in Form von Computerprogrammen können Universen von möglicherweise unbegrenzter Komplexität geschaffen werden. Außerdem [...] *handeln* die so formulierten und entwickelten System ihren eigenen Programmen gemäß."

Programmieren erscheint als Konstruieren handelnder Systeme, die – folgt man der Art und Weise, wie über sie geredet wird – auch eine Ausdehnung und körperlichen Charakter zu haben scheinen. In Kapitel 4 werde ich darauf, z. B. bei der Besprechung des „FORTRAN translators" (Abschnitt 4.4.2.3.2) oder des „logic theorist" (Abschnitt 4.6.1.2), noch zurückkommen. Wenn die reale Welt als „programmiert" erscheint und die Details der materiellen Körper hinter ihrer logischen Funktion verschwinden, können Programme als organismenähnliche, selbsttätig handelnde Entitäten gedacht werden.

3.2.2 Programmieren als behavioristisch-atomistische Technik

Zwar hatte von Neumann im wesentlichen die Programmierung von Berechnungen im Sinn. Seine 1948 in „The General and Logical Theory of Automata" entwickelte Sicht auf Organismen aus dem Blickwinkel der Rechenmaschine vermag jedoch – auch über den Bereich der Berechnungen hinaus – dazu zu dienen, die bei der Programmierung auf den zu programmierenden Gegenstandsbereich eingenommene Perspektive zu charakterisieren. Von Neumanns Sicht bietet zudem einen Hintergrund, vor dem Programme mal als Beschreibung, mal als operierende Entitäten aufgefaßt werden können und trägt damit dazu bei, zu erklären, warum die Grenze zwischen Organismus, Maschine und Symbolsystem im Diskurs der Programmierung so schwer zu fassen ist.

In dem Papier zur Automatentheorie wird deutlich, daß – wie ich in Abschnitt 3.1 aufgezeigt habe – aus der Perspektive der Rechenmaschine gesehen die Organismen die Struktur der Rechenmaschine widerspiegeln: Sie sollen untersucht werden als zusammengesetzt aus elementaren Stimulus-Response-Automatismen. Ebenso scheinen zeitliche Prozesse in diskrete Abschnitte unterteilbar zu sein. Aus der Sicht einer Maschine, die in einem bestimmten Zeittakt arbeitet, der die Operationen ihrer als Stimulus-Response-Automatismen beschreibbaren Bauteile synchronisiert, ist das nur konsequent. Was diese Maschine kann, ist, die elementaren Operationen ihrer elementaren Bausteine in einer bestimmten zeitlichen Anordnung auszuführen. Was programmiert werden soll, muß letztlich in diese elementaren Operationen zerlegt werden. In dem zu programmierenden Gegenstandsbereich werden deshalb zunächst bestimmte Einheiten isoliert und anschließend in bezug auf ihr Ein-/Ausgabeverhalten

sowie ihr Zusammenwirken eindeutig beschrieben, ähnlich wie von Neumann 1948 mit Hilfe der „axiomatic procedure“ Organismen als Strukturen von Stimulus-Response-Automatismen darstellen wollte. Christiane Floyd hat das 1996 in einem Artikel, in dem sie sich mit der „Methode der Informatik“ befaßt, wie folgt charakterisiert (S. 245): „Diskretisierung durch Zerlegung in trennbare Elemente und ihre Verknüpfung, wobei Gegenstände und Operationen identifiziert sowie die Regeln zu ihrer Verknüpfung herausgebildet werden“. In diesem Prozeß erlaubt der atomistische Blick, den Gegenstandsbereich in isolierbare Einheiten zu zerlegen, der behavioristische Blick, diese sowie ihr Zusammenwirken durch ihr Ein-/Ausgabeverhalten eindeutig zu beschreiben.

Bei der Programmierung werden die ausgemachten Operationen nun – von einem Programmierer oder automatisch von z. B. einem Compiler – in kleinere Verhaltenseinheiten aufgesplittet, bis diese von den elementaren Bauteilen der Rechenmaschine ausgeführt werden können. Auf den verschiedenen Verfeinerungsstufen scheinen die elementareren Verhaltenseinheiten das Verhalten der aus ihnen zusammengesetzten gröberen Einheiten hervorzubringen. Die elementareren Einheiten scheinen insofern beschreibenden wie auch operierenden Charakter zu haben. Einerseits beschreiben sie die allgemeineren Verhaltenseinheiten durch kleinere Verhaltenseinheiten; andererseits wird das gröbere durch das elementarere Verhalten scheinbar realisiert. Wenn in diesem Prozeß der Verfeinerung eine Stufe erreicht wird, auf der die jeweils elementareren Verhaltensformen von den in der Maschine verfügbaren Bauteilen ausgeführt werden können, kann das Programm in der Maschine tatsächlich operieren. An die Stelle der im zu programmierenden Gegenstandsbereich ausgemachten diskreten Elemente treten in der programmierten Maschine letztlich die elementaren Bauteile der Maschine, die im Zusammenspiel die gewünschten Operationen hervorbringen.

Das Programmieren setzt also eine Sicht auf den zu programmierenden Bereich als Automaten, als verknüpfte Struktur aus Ein-/Ausgabe-Automatismen, voraus und führt dieses Modell in einen technischen Automaten – die programmierte Maschine – über. Das Programmieren kann in dem Sinne als atomistische Technik verstanden werden, als es darauf beruht, zunächst den zu programmierenden Gegenstandsbereich in Einheiten und diese anschließend in Untereinheiten aufzusplitten usw. Es erscheint als behavioristische Technik insofern, als das Verhalten der jeweiligen Einheiten im Zentrum der Betrachtung steht. Dieses wird als eindeutig und vollständig beschreibbar und als durch untergeordnete – ebenfalls vollständig und eindeutig beschreibbare – Verhaltenseinheiten hervorgebracht aufgefaßt. Ein behavioristischer Handlungsbegriff, nach dem Handeln aus (regelhaften und vollständig beschreibbaren) Reiz-Reaktionsabläufen erklärbar ist, ermöglicht dabei, die verschiedenen Operationen auch

als Handlungen zu interpretieren. Die jeweils elementareren Verhaltenseinheiten stehen dabei für die Reiz-Reaktionsabläufe, die das Handeln hervorbringen.

Wie anhand des von Neumannschen Automatenbegriffs deutlich wurde, wird vor dem Hintergrund eines behavioristischen Handlungsbegriffs die Grenze zwischen technisch und natürlich sowie zwischen Organismus und Symbolsystem opak. Das gilt auch für Programme: Einerseits können sie als auf der Maschine realisiert und damit als maschinelle Entitäten verstanden werden; andererseits scheinen ihre Operationen vor dem Hintergrund eines behavioristischen Handlungsbegriffs dem menschlichen Handeln zumindest prinzipiell zu entsprechen. Schließlich können sie auch als Notationen gedacht werden, die beschreiben und Verhalten bzw. Handeln auch selbst hervorbringen. Wie Organismen in einer behavioristisch-atomistischen Sicht – ob der angenommenen vollständigen Beschreibbarkeit des Verhaltens ihrer Elemente – als agierende Symbolsysteme konzipiert werden können, werden Programme im Gegenzug als eine Art handelnde Organismen vorstellbar.

3.3 Zusammenfassung: Mensch – Maschine – Symbolsystem

In Kapitel 2 und 3 wurde deutlich, daß die Grenzen natürlich-technisch sowie (technisch konzipierter) Organismus-Symbolsystem in den hier besprochenen Texten verschwimmen. In diesem Abschnitt werde ich das noch einmal zusammenfassend darstellen und dabei auch die Rolle behavioristischer Vorstellungen herausarbeiten. Zudem zeige ich, daß die diesen Texten unterliegenden Orientierungsmuster auch für die Herausbildung der Wissenschaftsdisziplin Informatik von zentraler Bedeutung waren. Von Neumann ist nicht zuletzt in dem Sinne einer der Väter der Informatik, als in seinen Texten die speicherprogrammierbare Rechenmaschine und Programme in ein Sinngeflecht einbettet werden, das erlaubt, hier universellere als nur maschinelle Konstrukte zu sehen.

Wie ich gezeigt habe, wird die Rechenmaschine in den in Kapitel 2 und 3 besprochenen Texten anthropomorphisiert; das menschliche Nervensystem wirkt im Gegenzug technisch. Menschlicher und maschineller Rechner erscheinen so als ähnlich: beide erhalten Instruktionen, die sie im Gedächtnis ablegen, und die von dort den weiteren Verlauf einer Berechnung zu steuern scheinen. Deutlich wurde auch, daß die sich in den von Neumannschen Texten dokumentierende technisch geprägte Sicht auf Organismen im engen Zusammenhang mit anderen Arbeiten der Zeit steht: Sie wird auch in McCullochs und Pitts Arbeit von 1943 (vgl. Abschnitt 2.2.2) und Wieners Arbeiten (vgl. Abschnitt 2.1, 3.1.1) deutlich. In diesen Texten spielt ein behavioristischer Handlungsbegriff bei der Identifizierung technischer und menschlicher Tätigkeiten eine zentrale Rolle. Sinnhaftigkeit von Handeln ist – wie das für beha-

vioristische Untersuchungen von Handeln charakteristisch ist – weder für McCulloch und Pitts, noch für Rosenbluth, Wiener und Bigelow oder von Neumann ein Thema. Trotzdem sind sie der Meinung, das Entstehen von Denken und Handeln im wesentlichen erklärt zu haben:

- Nach McCullochs und Pitts Theorie sind die zukünftigen Aktionen des Nervensystems durch die Struktur des Netzes immer schon festgelegt, während man auf vergangene Zustände nicht rückschließen kann (McCulloch, Pitts 1943, S. 25):

  „Causality [...] has [...] never [...] been as irreciprocal as in this theory. Specification for any one time of afferent stimulation and of the activity of all constituent neurons, each an ‚all-or-none' affair, determines the state. Specification of the nervous net provides the law of necessary connection whereby one can compute from the description of any state that of the succeeding state, but the inclusion of disjunctive relations prevents complete determination of the one before."

  Verhalten folgt nach ihrer Vorstellung den im Netz inkorporierten Regeln, der Zweck einer Handlung spiegelt sich im Aufbau des Netzes selbst. Trotz dieser Beschränkung des Handlungsbegriffs auf das Ausführen von vorgegebenen Regeln folgern sie, daß die gesamte Psychologie als zweiwertige Logik betrachtet werden kann (vgl. S. 25).
- Rosenbluth, Wiener und Bigelow (1943) geht es zwar darum, die Wichtigkeit des Zweckbegriffs zu unterstreichen, jedoch reduzieren sie Zweck auf das Erreichen einer bestimmten zeitlich-räumlichen Korrelation zweier Objekte. Warum diese Korrelation erreicht werden soll, welcher Sinn damit verbunden wird und welche die treibende Kraft ist, die diese Korrelation anstrebt, war nicht Teil ihrer Überlegungen. Trotz dieser Beschränkung auf sehr spezielle Formen des Verhaltens stellen sie fest, daß es qualitative Unterschiede des Verhaltens zwischen lebenden Organismen und Maschinen nicht gibt.
- In von Neumanns Texten schließlich ist die Frage danach zentral, wie sein Rechengerät die „Befehle", die es erhält und in seinem Speicher quasi verinnerlicht, selbständig und ohne weitere intelligente menschliche Intervention ausführen kann. Es interessiert also die Frage danach, wie man das selbsttätige Ausführen einer regelgeleiteten Handlung, die einem von außen gesetzten Ziel dient, mit einer Maschine umsetzen kann. Trotz der Beschränkung auf dieses Verhaltenssegment ging von Neumann davon aus, daß menschliche Neuronen und Vakuumröhren und sein als Organismus beschriebenes Gerät und das menschliche Nervensystem zumindest ähnlich funktionieren.

Das Entstehen von Zielen und Zwecken, von Sinnhaftigkeit des Handelns, wird also von den Autoren nicht problematisiert. Es wird deutlich, daß sie auf einen behavioristischen Handlungsbegriff rekurrieren, nach dem Handeln aus dem Ablauf von Reiz und Reaktion erklärbar ist. Organismen und das Nervensystem können so als Regeln befolgende Mechanismen gesehen werden, die auf einen bestimmten Stimulus mit einer bestimmten Response reagieren. Ihr Verhalten ist so gesehen vollständig und

eindeutig beschreibbar und durch aus der Umwelt eingehende Reize eindeutig bestimmt. Bereits Geschehenes – das, was den Sinn einer Handlung konstituieren könnte – ist in dieser Sicht für den weiteren Fortgang der Handlung nicht von Bedeutung. 1943 formulierten McCulloch und Pitts diese Sicht auf menschliches Denken besonders prägnant (S. 27): „The psychiatrist may take comfort from the obvious conclusion concerning causality – that for prognosis, history is never necessary." Organismen auf Grundlage eines behavioristischen Handlungsbegriffs als regelhaft funktionierend vorzustellen legt nahe, daß sie zumindest prinzipiell in Maschinenform nachgebaut werden können – eine Vorstellung, die bei allen Autoren deutlich wird.

Die Herausbildung eines behavioristischen Handlungsbegriffs und damit zusammenhängend die Analogie Nervensystem-Rechenmaschine muß in einem breiteren Kontext verstanden werden. Beides kann auch als Ausdruck einer Tradition der neuzeitlichen Naturwissenschaft gesehen werden, in der Naturprozesse als mechanisch-technisch konstruiert werden. Zu diesem Themenbereich schiebe ich an dieser Stelle nur einige kurze Bemerkungen ein und verweise für eine ausführlichere Behandlung auf Scheich (1998) und Kutschmann (1986).

Das Naturwissenschaftsideal des 19. Jahrhunderts etikettierte menschliches Beobachten und Sinneserfahrung als Feinde der Erkenntnis. Der mechanische Apparat, als unbeeinflußt von jeglichen Vorannahmen angesehen, galt als Ideal des experimentellen Beobachtens. In den mechanisch erzeugten Darstellungen der Natur sollte diese für sich selbst sprechen (vgl. dazu Scheich 1999). Das Mißtrauen gegenüber der eigenen sinnlichen Wahrnehmung verdrängte den Körper des Wissenschaftlers als Sitz der Erkenntnis (Kutschmann 1986, S. 396):

> „Der Naturwissenschaftler der Neuzeit ist, im Idealfall zumindest, nicht mehr körperlich in den Prozeß der Naturaufklärung involviert; vielmehr erfährt und notiert er die Natur aus einer Distanz heraus, die ihn weder körperlich-physiologisch noch leiblich existentiell betroffen sein läßt."

Im 20. Jahrhundert treten die Sinne als Instanz der Realitätserfahrung weiter in den Hintergrund zugunsten einer Theoriebildung, die Natur vermittelt durch apparatetechnische Experimente erfährt. Die künstlich hergestellten Naturprozesse werden zunehmend als identisch mit dem Original wahrgenommen (Scheich 1998, S. 13):

> „Bereits mit der Mechanisierung von Objektivität wurde eine Dynamik in Gang gesetzt, die für die Wissenschaftsentwicklung des 20. Jahrhunderts wesentlich werden sollte: Die Identifizierung von Objekt und Repräsentation, die Gleichsetzung von Bild und Wirklichkeit. [...] Schritt für Schritt wird der Unterschied zwischen dem Objekt und seinem Abbild nivelliert."

Naturprozesse werden demnach in den Naturwissenschaften technisch vermittelt wissenschaftlich wahrgenommen. Daß Naturprozesse und Organismen in den von mir besprochenen Texten aus einer technischen Perspektive interpretiert werden, ist also

auch Ausdruck eines wissenschaftlichen Erkenntnisideals, das für die neuzeitlichen Naturwissenschaften typisch und grundlegend ist.

Die in dem letzten Zitat angesprochene Identifizierung von Objekt und Abbild, die in der Folge der technisch vermittelten Wahrnehmung von Natur entsteht, führt zu der zweiten oben angesprochenen Grenze zwischen Organismus und Symbolsystem. Daß die Mechanisierung von Objektivität und damit verbunden die technische Wahrnehmung der Natur dazu führt, den Unterschied zwischen Bild und Wirklichkeit Schritt für Schritt zu nivellieren, deutet sich auch in von Neumanns Texten an: Das technisch konzipierte Nervensystem erscheint in „The General and Logical Theory of Automata" auf Grundlage der „axiomatic procedure" – wie schon das Rechengerät des „First Draft" – als Muster aus Stimulus-Response-Automatismen, deren Verhalten vollständig und eindeutig beschreibbar scheint. Auch diese Sicht des Nervensystems weist Ähnlichkeit zu behavioristischen Vorstellungen auf: An die Stelle der „Reflexe", die – in Muskeln und Nervensystem verankert – das Verhalten bewirken, treten hier die für die konstitutiven Elemente stehenden Stimulus-Response-Automatismen, die im Zusammenspiel das Verhalten des Organismus konstituieren. Die Materialität des Nervensystems (bzw. die Körperlichkeit des ganzen Menschen, von dem das Nervensystem eine Teil ist) tritt hinter seine aus dieser Perspektive scheinbar mathematisch-logisch beschreibbare Struktur zurück. Aus einer behavioristisch inspirierten mathematischen Forscherperspektive kann das Nervensystem so als Symbolsystem konzipiert werden. Seine Organisation und sein Funktionieren als Gesamtes zu untersuchen erscheint, als Untersuchung eines Musters und kann – ähnlich wie Rechenmaschinenbau in den in Kapitel 2 besprochenen Texten – als Gegenstandsbereich der Mathematik und Logik gesehen werden.

Beide – die Rechenmaschine und das Nervensystem – können auf Grundlage der „axiomatic procedure" in ähnlicher Weise mathematisch-logisch beschrieben und wissenschaftlich untersucht werden; und diese Untersuchung scheint sich auf einer höheren Allgemeinheitsebene zu bewegen, als nur eine Maschine zu betreffen. Sie scheint Gültigkeit sowohl im technischen wie auch bis zu einem gewissen Grad im biologischen Bereich – für das Nervensystem – beanspruchen zu können. Wie in den in Kapitel 2 besprochenen Texten Rechenmaschine und Code erscheint in dem Papier zu Automaten auch das Nervensystem als Hybridobjekt. Was Keller für den Körper in der modernen Biologie feststellt – daß er mal als Maschine, dann als Nachricht erscheint (vgl. Keller 1995, S. 118)[84] –, gilt ähnlich für die von Neumannsche Konzeption des Nervensystems: Es erscheint als technischer, mechanischer Gegenstand und wird auch als Symbolsystem gedacht.

---

84 Zum Verhältnis Genetik-Programmierung s. Abschnitt 3.2.1.

Zu Beginn des zweiten Kapitel habe ich die These aufgestellt, daß von Neumann nicht nur in einem technischen Sinne für die Entwicklung der Rechenmaschinen und ihrer Programmierung wichtig war, sondern besonders auch dafür, wie die Rechenmaschine und ihre Programmierung in der Folgezeit wahrgenommen und in Sinnzusammenhänge eingebettet wurde. Wäre der erfolgreiche Bau von Maschinen das Erfolgskriterium, das von Neumann als einen – wenn nicht den – Vater der Informatik erscheinen läßt, würde von Neumann diese herausragende Bedeutung heute wohl kaum zugesprochen. Denn die ersten speicherprogrammierbaren Rechenmaschinen wurden von anderen gebaut, die – besonders auch aus dem Bereich der Radartechnologie – versiertes technologisches Know-how mitbrachten (vgl. Abschnitt 2.3).

Warum also wird heute von der „Von-Neumann-Architektur" als dem Konzept gesprochen, das den weitaus meisten speicherprogrammierbaren Rechenmaschinen unterliegt? Furger und Heintz (1998) weisen darauf hin, daß die Von-Neumann-Architektur sich zwar Anfang der 50er Jahre als eine Art Leitbild durchgesetzt hatte und auch normierend wirkte. Trotzdem waren in konkreten Rechnerbauprojekten wie dem ERMETH-Projekt in Zürich, das sie in ihrem Artikel näher untersuchen, die Entscheidungsspielräume groß und der tatsächliche Aufbau des Rechners stark von lokalen Gegebenheiten und Bedürfnissen geprägt. Furger und Heintz folgern,

> „daß (technologische) Paradigmen [wie die Von-Neumann-Architektur] ihre Bedeutung erst im Nachhinein erhalten - als retrospektive Rationalisierungen situativ bedingter Entscheidungen. [...] Erst von heute aus gesehen bekommt die Von-Neumann-Architektur ihre Funktion: sie dient dazu, die vielen situativ und ad-hoc getroffenen Entscheidungen zu linearisieren und ihnen eine Plausibilitätsstruktur zu geben" (S. 234).

So gesehen hat die Von-Neumann-Architektur weniger in dem Sinne Bedeutung, als sie in einzelnen Rechnerbauprojekten tatsächlich das Vorbild bot, sondern vielmehr deshalb, weil sie ex post dazu dienen kann, die ehemals getroffenen technischen Entscheidungen zu rationalisieren. Trotzdem bleibt die Frage, warum sich die Von-Neumann-Architektur zumindest in dem Sinne durchgesetzt hat, daß sie heute dazu benutzt werden kann, die ehemals getroffenen Entscheidungen zu plausibilisieren. Eine große Rolle spielt dafür sicherlich die große publizistische Aktivität von Neumanns und des IAS-Instituts, welche die hier entwickelten Ideen bekannt machte. So macht Heintz darauf aufmerksam, daß für die Durchsetzung der Von-Neumann-Architektur nicht nur technische Faktoren, sondern vor allem auch diskursive Kompetenzen entscheidend waren (vgl. Heintz 1993, S. 226f.). Zudem spielen ihr zufolge auch unterliegende Deutungssysteme – in meiner Terminologie Sinnstrukturen – bei der Frage danach, welche technischen Varianten sich durchsetzen, eine große Rolle (Heintz 1993, S. 228):

> „Zu welcher Beurteilung man gelangt, welche technischen Varianten seligiert werden (und welche in Vergessenheit geraten), ist auch abhängig von den Deutungssystemen, an denen man sich orientiert, und von der Struktur des kommunikativen Prozesses, in dessen Verlauf sich eine gemeinsame Sicht herauskristallisiert."

Heintz untersucht im Anschluß die „kulturelle Erfindung des Computers“ und stellt fest, daß für „die damaligen Computerpioniere [...] der Computer eine gigantische Rechenmaschine“ (S. 231) war. Erst später hätte man begonnen, ihn „nun mit ‚Gehirn‘ und ‚Denken‘ zu assoziieren“ (S. 232).

Nun war für von Neumann die Analogie Rechenmaschine-Nervensystem/Gehirn von Anfang an ein zentraler Punkt. Seine Texte behandeln zwar ein Rechensystem; der McCulloch/Pitts-Text (1943), aus dem von Neumann seine Vorstellung des Nervensystems ableitet, entwirft jedoch ein Bild des Denkens, nach dem dieses sich weitgehend in Rechnen erschöpft. Auch wenn die Maschine „nur“ rechnet, kann sie so als elektronisches Nervensystem und Gehirn gesehen werden. Umgekehrt setzt bereits die Deutung, daß die Maschine „rechnet“, eine gewisse Ähnlichkeit des menschlichen Gehirns mit der Maschine voraus. Eine Sicht auf die Rechenmaschine, nach der diese mit dem Nervensystem und Gehirn verglichen werden kann, war also vom Beginn der Speicherprogrammierung an – zumindest in von Neumanns Schriften – aktuell.

Ein entscheidender Punkt, warum heute von Neumann so oft als Vater des modernen Computers und der Informatik gesehen wird, ergibt sich, so denke ich, aus dieser Deutung der Rechenmaschine als nervensystemartiger Organismus. Zumindest für die Herausbildung der Wissenschaft Informatik spielte sie eine zentrale Rolle – und damit auch dafür, daß ex post die Von-Neumann-Architektur von Informatikern als Paradigma des Rechenmaschinenbaus gesehen werden konnte. Von Neumanns Arbeiten zeigen, wie ein technischer Blick auf Organismen und das Nervensystem erlaubt, diese und die Rechenmaschine als gleichartige Symbolsysteme zu konzipieren. Beiden scheinen so mit den gleichen Methoden untersuchbar; die Ergebnisse, die auf Grundlage der Untersuchung der Symbolsysteme erzielt werden, scheinen von größerer Allgemeinheit zu sein, als nur über eine Maschine etwas auszusagen. Ein solches Vorgehen erlaubt, die Informatik als eine Wissenschaft zu sehen, in der es nicht um eine Maschine und ihrer Programmierung, sondern um Informationsverarbeitung allgemein geht,[85] was ich nun kurz aufzeige.

In einer Untersuchung zur Entstehung der Informatik als wissenschaftlicher Disziplin stellt Eulenhöfer nach einer Analyse dreier Texte[86], die in der Anfangszeit der Informatik von prominenten Protagonisten dieser Disziplin zum Thema „Was ist Informatik“ verfaßt wurden, fest (Eulenhöfer 1998b, S. 219):

---

85 Neben von Neumann wird auch Turing häufig unter Titeln wie „Vater des Computers“ geführt. In seinem Papier von 1936 zieht Turing, ähnlich wie von Neumann, Parallelen zwischen seinen Maschinen und menschlichen Rechnern. Letztlich handelt es sich bei Turings Maschinen um immaterielle Maschinen, also Symbolsysteme. Daß gerade diese beiden so häufig als Urväter der Computer oder Informatik genannt werden, verweist so darauf, daß ihren Texten bereits sehr früh die heute in der Informatik aktuelle Sicht auf Rechenmaschinen als mathematische Artefakte unterliegt.

86 Dabei handelt es sich um (Bauer 1974), (Brauer 1973), (Zemanek 1971).

„Die Gerätetechnik wird [...] in den Texten auf doppelte Weise aus dem Gegenstandsbereich der Informatik eliminiert. Zum einen erscheinen die Computer und Informationsverarbeitungsanlagen nicht in ihrer Materialität, zum anderen scheinen sie aber auch grundsätzlich für das dynamische Verständnis von Informationsverarbeitung und von Programmen nebensächlich zu sein.

Auf diese Weise präsentieren die drei Autoren die Informatik letztlich als eine Wissenschaft, die sich ausschließlich mit Notationen und der Transformation von Notationen befaßt."

Es sind also nicht die Maschinen, die den Gegenstand der Informatik bildet, sondern Notationen, also Symbolsysteme. Wie die Maschine als Symbolsystem konzipiert werden kann, hatte von Neumann in seinen Texten vorgeführt. Gleichzeitig hatte er über die Analogie Nervensystem-Rechenmaschine die Möglichkeit eröffnet, solchen Symbolsystemen auch Aussagekraft in bezug auf menschliches Denken zuzusprechen. Der sich daraus ergebende Anspruch auf allgemeingültige Aussagen im Bereich der Informationsverarbeitung insgesamt durchzieht – häufig diffus – viele Schriften der Informatik. So bezeichnet Bauer (1974) – ein zentraler Akteur im Prozeß der Institutionalisierung der bundesdeutschen Informatik – die Linguistik, Biologie, Physiologie, Psychologie und Psychiatrie als Disziplinen, die eine „geistige Querverbindung" zur Informatik besitzen. In seinem 1973 erschienenen, zusammen mit Goos verfaßten Buch „Informatik. Eine einführende Übersicht" werden entsprechend in dem ersten Kapitel „Information und Nachricht" die „Sinnesorgane" (Überschrift zu 1.2), die „Arbeitsweise der Sinnesorgane, Reizleitung" (Überschrift zu 1.2.1) und die „Reizverarbeitung im Gehirn" (Überschrift zu 1.2.2) beschrieben. In 1.2.3 folgern die Autoren, auch wenn sie zur Vorsicht bei Aussagen über menschliche geistige Funktionen warnen (S. 14):

„Der Ausblick in die Sinnesphysiologie und -psychologie lehrt uns verschiedenes: Zunächst, daß Informationsverarbeitung (und nicht nur Informationsübertragung) ein unabdingbarer Bestandteil unserer Sinneswahrnehmung ist. Wenn wir im weiteren von künstlicher, nämlich von Menschen erfundener und maschinell ausgeführter Informationsverarbeitung sprechen, so soll uns das daran erinnern, daß Informationsverarbeitung nichts grundsätzlich Neues ist."

Ähnlich spricht auch heute noch der aktuelle bundesdeutsche Forschungsführer Informatik dem „Gebiet der Informatik universellen Charakter" zu. Daß hier die Informatik „durch die drei Begriffe Intelligenz – Formalismen – Technik oder als Intelligenzformalisierungstechnik" (Brauer, Münch 1996, S. 13) gekennzeichnet wird, setzt die Art und Weise fort, wie von Neumann 1945 den Rechner zu einem universelleren Ding als eine Maschine machte: Er beschrieb eine Maschine – Technik –, die er mit Hilfe neurophysiologischer Überlegungen als Organismus – eine Art Intelligenz – charakterisierte und die auf Grund mathematischer Argumentationen und Stilmittel als eine jenseits des Materiellen liegende allgemeingültige Struktur – als eine Art Formalismus – erscheint. Gemeinsam ist dem Forschungsführer und den von Neumannschen Texten die Hybridobjekt-Sichtweise, in der die Maschine als Organismus, der Organismus als maschinell und Maschine wie Organismen als Symbolsysteme gesehen werden können.

Kapitel 4: Programmiersprachen

Während der Terminus „Sprache" in der Programmierung in den 40er und Anfang der 50er Jahre nur sporadisch verwendet wurde, wurde es Mitte der 50er Jahre gängig, über „Sprachen" als Mittel der Kodierung zu reden. Aus dem zunächst metaphorischen Gebrauch des Sprachbegriffs wurde der etablierte Fachterminus „Programmiersprache". Daß sich diese terminologische Verschiebung unreflektiert und weitgehend unbemerkt vollzog, zeigen die zeitgenössischen Fachliteratur über die ersten großen und teilweise bis heute weit verbreiteten „Programmiersprachen" FORTRAN, IPL und LISP, ALGOL und COBOL[87] sowie Interviews, die ich mit an diesen Sprachentwicklungen Beteiligten führte.

Die beteiligten Wissenschaftlerinnen und Wissenschaftler wechselten also wie selbstverständlich zum Sprachbegriff. Die neue Sichtweise auf die Kodiersysteme als Sprachen blieb nicht folgenlos für die weitere Entwicklung. Mit der Durchsetzung des Sprachbegriffs bekamen die ehemaligen Codiersysteme eine „Syntax", eine „Semantik" und eine „Grammatik". Nicht zuletzt entstand im Zuge der neuen Terminologie das neue informatische Forschungsfeld der formalen Semantik. Da die Verwendung des Sprachbegriffs im Diskurs der Programmierung eine wichtige Rolle spielte und die Technik- und Wissenschaftsentwicklung in diesem Bereich orientierte, stelle ich in Abschnitt 4.1 einige Überlegungen zu dem unterliegenden Sprachbegriff an: Welche Sicht auf Sprache ermöglichte, daß künstlich hergestellte Kodiersysteme als Sprachen gedacht werden konnten? Um die Herausbildung von Programmiersprachen nicht nur ideengeschichtlich, sondern auch zeitgeschichtlich in einen breiteren Kontext einzubetten, gebe ich in Abschnitt 4.2 anschließend eine kurze Darstellung der Situation der Wissenschaft und des Computereinsatzes in den USA der Nachkriegszeit. In 4.3 stelle ich überblicksartig den Stand der Technik im Bereich der Kodiersysteme dar, bevor ich darauf aufbauend auf die Entwicklung der ersten großen Programmiersprachen FORTRAN, ALGOL und COBOL, LISP und dessen Vorläufer IPL eingehe.

FORTRAN, ALGOL 60, COBOL und LISP gelten in der Entwicklung von Programmiersprachen als Meilensteine (Wegner 1976, S. 1208f.):

> „The four most important milestones [in the development of programming languages] are probably the following ones.
>
> *Fortran*, which provided an existence proof for higher level languages, and is still one of the most widely used programming languages.
>
> *Algol 60*, whose clean design and specification served as an inspiration for the development of a discipline of programming languages.

---

87 FORTRAN: Formula Translating System (Entwicklungsbeginn 1954, USA); IPL: Information Processing Language (Entwicklungsbeginn 1956, USA); LISP: List Processor (Entwicklungsbeginn 1958, USA); ALGOL: Algorithmic Language (Entwicklungsbeginn 1955, international); COBOL: Common Business Language (Entwicklungsbeginn 1959, USA).

Cobol, which pioneered the development of data description facilities, was adopted as a required language on department of defense computers and has become the most widely used language of the 1970's.

Lisp, whose unique blend of simplicity and power have caused it to become both the most widely used language in artificial intelligence and the starting point for the development of a mathematical theory of computation. [...]

Fortran is perhaps the single most important milestone in the development of programming languages."

FORTRAN, ALGOL 60, COBOL und LISP waren also nicht nur technische Neuerungen, sondern auch Ausgangspunkt oder zumindest wichtig für weitere, für die Herausbildung der „Computer Science" bzw. Informatik zentrale Entwicklungen wie den Compilerbau, die Theorie der Programmiersprachen sowie die formale Semantik, Datenbanken und Künstliche Intelligenz. Alle vier Sprachen wurden breit eingesetzt und spielen auch heute noch – besonders FORTRAN und COBOL – in der Programmierung eine zentrale Rolle. Ich werde in diesem Kapitel die Orientierungsmuster, die bei ihrer Entstehung zentral waren, anhand der ersten wichtigen Publikationen und Interviews mit beteiligten Akteuren herausarbeiten. Im Sinne einer komparativen Analyse werde ich dabei – neben einer kurzen Darstellung des jeweiligen Entwicklungskontextes – vor allem die ersten Texte, in denen die jeweiligen Akteure ihre Entwicklung der Öffentlichkeit präsentierten, besprechen und so Gemeinsamkeiten und Unterschiede aufzeigen.

Besonders ausführlich behandele ich die Entwicklung von FORTRAN 1 (Abschnitt 4.4), in deren Rahmen der Sprachbegriff zum ersten Mal durchgängig und systematisch verwendet wurde. Das Programmieren mit der als mathematikähnlich gesehenen FORTRAN-Sprache schien im wesentlichen der von Neumannschen Phase der mathematischen Problemvorbereitung zuzurechnen zu sein – und damit dem Arbeitsgebiet der Wissenschaftler und Ingenieure, die diese erstellten, und nicht mehr dem der auf diesen Vorarbeiten aufsetzenden und speziell ausgebildeten Kodierer (vgl. Abschnitt 2.4.2). Das Kodieren sollte mit dem „FORTRAN Automatic Coding System" der „translator" übernehmen, der den in der FORTRAN-Sprache erstellten „generellen Plan" in Maschinensprache übersetzte. FORTRAN sollte ermöglichen, daß die Maschine sich selbst kodiert.

ALGOL und COBOL werde ich nur knapp untersuchen in bezug auf die Frage, wie sich die Verwendung des Sprachbegriffs mit dem Hauptanliegen dieser beiden Programmiersprachenentwicklungen, rechnerunabhängige Sprachen bereitzustellen, veränderte (Abschnitt 4.5). Dabei zeigt sich, daß beide Sprachen nicht nur für die Programmierung, sondern auch für die Verständigung zwischen Menschen entwickelt wurden, und – besonders COBOL – in einem recht buchstäblichen Sinne als Sprachen verstanden wurden.

Bei der Besprechung von LISP gehe ich zunächst auf IPL (Abschnitt 4.6.1), den wichtigsten Vorläufer von LISP, ein. Im Rahmen der IPL-Entwicklung wurde – ange-

lehnt an Vorstellungen über die menschliche Gedächtnisfunktion – das Listenkonzept entworfen und implementiert, das später auch in LISP eine zentrale Rolle spielte. Mit Hilfe von Listen und heuristischer Programmierung schien der Computer ähnlich flexibel wie ein Mensch Probleme lösen zu können. Der Computer erschien als eine Art Labor, in dem Informationsverarbeitungssysteme allgemein empirisch untersucht werden können. Die Programme, mit deren Hilfe ein Informationssystem realisiert wurde, wurden gleichzeitig als Theorie über menschliches Problemlösen gesehen.

Ähnlich wie in IPL sollte auch mit Hilfe von LISP (Abschnitt 4.6.2) menschliches Denken und besonders gesunder Menschenverstand programmiert werden. Zudem sollte LISP der eleganten Programmierung von Berechnungen dienen – menschliches Denken und Berechnen schienen keinen großen Unterschied zu machen. Neben der Programmierung sollte LISP auch der Kommunikation mit LISP-Programmen dienen. Damit sollte erreicht werden, daß LISP-Programme sich selbst im „Gespräch" mit dem Menschen verbessern konnten. Angestrebt wurde, daß Programme sich selbst programmieren konnten. Die Gestalt der LISP-Sprache leitete McCarthy 1958 aus mathematisch-formalen Überlegungen ab – LISP wird hier als Berechenbarkeitsformalismus dargestellt. LISP erscheint so als universale Sprache, die der Programmierung von Denken und Berechnungen, der Kommunikation mit LISP-Programmen und der mathematischen Theoriebildung dienen kann.

In der abschließenden Zusammenfassung (Abschnitt 4.7) argumentiere ich, daß den besprochenen Programmiersprachenentwicklungen ein Verständnis von Sprache unterliegt, nach der diese auf Grundlage von Regeln mechanisch hergestellt werden kann. Die Rechenmaschine konnte so als Sprachmaschine – oder wie es normalerweise heißt: als Informationsverarbeitungssystem – gesehen werden. Programmiersprachen – verstanden als Sprachen – wurden theoriebildend in verschiedensten Bereichen. Programme erschienen einerseits als beschreibende Texte, andererseits als operierend und handelnd. Ihr technischer Charakter spiegelt sich nun vor allem darin wider, daß ihnen operative Kraft zugesprochen wird. Verstanden durch die Folie menschlicher Tätigkeiten erscheinen sie auch als selbständig handelnd. Deutlich wird auch hier die Hybridobjekt-Sichtweise, in der Differenzen zwischen natürlich, technisch und Symbolsystem verschwimmen.

4.1 Herstellbare Sprachen

Der Sprachbegriff wurde – wie ich oben bereits bemerkt habe – in den Bereich der Programmierung übernommen, ohne daß dies von den Akteuren thematisiert wurde. Wie läßt sich diese Selbstverständlichkeit, mit der der Sprachbegriff in den Bereich der Kodiersysteme übertragen wurde, erklären? War es die immer größer werdende

Komplexität der damaligen Kodiersysteme, die diesen Übergang zur Sprachmetaphorik motivierte? Das mag sein. Ebenso war es jedoch – wie ich in diesem Abschnitt argumentiere – der Sprachbegriff der Zeit, der zumindest nahelegte, Sprache und Kodiersystem gleichzusetzen. Die Art und Weise, wie Sprache in dieser Zeit gedacht und verstanden wurde, prägte das Nachdenken über die Programmierung und erlaubte, das Programmieren als sprachliche Tätigkeit zu sehen. Die folgenden Überlegungen zum Sprachbegriff lehnen sich eng an Foucaults „Die Ordnung der Dinge" an und können auch als Beispiel dafür gesehen werden, wie die zu einer bestimmten Zeit in einer Gesellschaft übliche Begrifflichkeit die Technik- und Wissenschaftsentwicklung dieser Zeit orientiert und ausformt.

Zwischen Kodiersystemen und Sprache besteht auf den ersten Blick ein zentraler Unterschied: Während Programmiersprachen offensichtlich herstellbar sind, gilt das für die natürliche Sprache, die den Einzelnen als „gewachsen" gegenübertritt, augenscheinlich nicht. In den Sprachwissenschaften des 19. Jahrhunderts etablierte sich jedoch ein Verständnis von Sprache, nach dem diese nicht mehr durch ihren Bezug auf die Dinge der Natur bestimmt ist, sondern einen von diesen abgesonderten, eigenständigen Gegenstand darstellt.

Noch im 16. Jahrhundert wurde Sprache begriffen als von Gott in der Welt niedergelegt. Als eine Sache der Natur mußte sie untersucht werden. Mit den Dingen der Welt verband sie eine von Gott gestiftete Analogiebeziehung, so daß Worte und Dinge nicht getrennt voneinander betrachtet werden konnten. Beginnend mit dem 17. Jahrhundert wurde diese tiefe Zusammengehörigkeit der Sprache und der Welt aufgelöst. Denkbar wird Sprache als ein abgesonderter Gegenstand, der von Menschen kritisiert, modifiziert und schließlich auch hergestellt werden kann. Wäre die Sprache der Moderne eine immer schon gegebene, von den Dingen der Natur nicht zu trennende Sache, wäre die Idee der Herstellung von Sprache durch den Menschen und damit eine „Programmiersprache" undenkbar gewesen.

Bis zum 19. Jahrhundert begründete sich der Wert der Sprache in der Tiefe, die sie zu gestalten erlaubte (Foucault 1974, S. 114):

> „Sprache repräsentiert das Denken, wie sich das Denken selbst repräsentiert. [...] Sie ist keine äußere Wirkung des Denkens, sondern selbst Denken. Dadurch wird die Sprache, wenigstens beinahe, unsichtbar. Es gibt für die Bildung der Sprache [...] keinen wesentlichen oder primitiven Akt der Bedeutung".

Sprache erschien als wohlgestaltet, wenn sie zuließ, das Denken angemessen zu strukturieren (Foucault 1974, S. 97):

> „Das Wissen [...] muß eine Sprache herstellen, die wohlgestaltet ist, das heißt, daß sie analysierend und kombinierend, wirklich die Sprache des Rechnens (*langue des calculs*) ist.[...] Es rückt alles Wissen in die Nähe einer Sprache und bemüht sich, allen Sprachen ein System künstlicher Symbole und Operationen logischer Natur zu substituieren."

Das Vorhaben von Leibniz, eine Erkenntnistheorie, eine universale logische Sprache und eine Rechenmaschine zu entwickeln, ist der vielleicht deutlichste Ausdruck dieses

Blicks auf die Sprache. Daß dieses Bild der Sprache auch später in der Informatik noch eine Rolle spielte, zeigen Geschichtskonstruktionen zur Informatik, die Leibniz bei der Entstehung der Informatik eine entscheidende Rolle zusprechen (vgl. z. B. Bauer 1974).

Im 19. Jahrhundert hörten die Wörter auf, die Erkenntnis der Dinge auf mehr oder weniger vorgegebene Weise zu rastern. Das Wort funktionierte jetzt nicht mehr, indem es die Wahrnehmung der Dinge mehr oder weniger adäquat übersetzte. Damit es das sagen konnte, was es sagte, mußte es nun Teil eines grammatikalischen Systems sein, welches das Wort determiniert. Sprache verstanden als ein System, das so aus sich heraus als eigenständige Einheit existiert, läßt zu, die Idee der Herstellbarkeit von Sprache im Sprachbegriff mitzudenken. Am Beginn des 20. Jahrhunderts entstanden entsprechend eine Reihe von Kunstsprachen wie z. B. Esperanto (Mehrtens 1990, S. 527):

> „In den Jahrzehnten um die Jahrhundertwende wurden so viele Pläne für künstliche Sprachen gemacht wie zu keiner anderen Zeit. Das zeitgenössische Standardwerk referiert für die Zeit von 1880 bis 1907 vierundvierzig Systeme."

Die Herstellbarkeit von Sprache stand also in dieser Zeit nicht mehr in Frage. Die später von Menschen hergestellten Kodiersysteme konnten so, ohne daß das als außergewöhnlich empfunden wurde, als „Sprachen" gedacht werden.

In der Zeit der Entstehung der ersten Programmiersprachen entwickelte Noam Chomsky sein Konzept der generativen Grammatiken (vgl. Chomsky 1956, 1957), die später in der Theorie der formalen Sprachen und damit auch in der Informatik eine große Rolle spielten. Chomsky arbeitete nach 1955 am „Research Laboratory of Electronics" am MIT, wo nach 1958 auch LISP entwickelt wurde; seine Ideen können von daher nicht unabhängig von der Diskussion um die Programmierung von Computern gesehen werden. Der Ausgangspunkt für Chomskys Arbeiten war, wie ich im folgenden – allerdings ohne dabei in die Tiefe zu gehen – aufzeige, die von Foucault beschriebene Idee von Sprache, nach der diese auf Grundlage von Regeln funktioniert, die in der Sprache selbst liegen.

Chomsky faßte Sprache 1956 wie folgt (Chomsky 1956, S. 13): „From now on I will consider a *language* to be a set (finite or infinite) of sentences, each finite in length and constructed out of a finite set of elements." Eine Sprache erscheint hier als Menge von klar unterscheidbaren Elementen – Sätzen –, die ihrerseits aus einer endlichen Menge klar unterscheidbarer Elemente aufgebaut sind. Deutlich wird hier ein atomistischer Blick auf Sprache, ähnlich wie ihn von Neumann 1948 auf Organismen, Maschinen und zeitliche Prozesse angewandt hatte (vgl. Abschnitt 3. 1.1). Aus dieser Perspektive gesehen kann eine Sprache auf Grundlage einer endlichen Grammatik erzeugt werden. Dabei faßt Chomsky eine Grammatik als „device", als Gerät, das die in Frage stehende Sprache erzeugt (Chomsky 1956, S. 18):

„Assuming the set of grammatical sentences of English to be given, we now ask what sort of device can produce this set (equivalently, what sort of theory gives an adequate account of the structure of this set of utterances). [...] We ask what sort of grammar is necessary to generate all the sequences of morphemes (or words) that constitute grammatical English sentences and only these. One requirement that a grammar must certainly meet is that it be finite."[88]

Chomsky versucht also, um die grammatikalisch korrekten Sätze der englischen Sprache zu beschreiben, ein Gerät anzugeben, das die möglichen Sätze erzeugt, oder auch eine Grammatik zu beschreiben, die diese Sätze generiert. Das Gerät wird dabei als Äqivalent einer Theorie gesehen, welche die Struktur der in der Sprache möglichen Sätze beschreibt. In Chomsyks Darstellung einer Grammatik dokumentiert sich die Hybridobjekt-Sichtweise: In obigem Zitat erscheint eine Grammatik als ein produzierendes Gerät und auch als eine Theorie. Zudem stellte Chomsky sich eine Grammatik auch als organisch im Menschen realisiert vor (Dittrich 1989, S. 9):[89]

„Das Verfügen über Sprache, die Kompetenz, in einer Sprache Neues ausdrücken zu können, bildet für ihn ein kognitives System oder ‚Organ'. Dieses kognitive System besteht aus den Regeln, mit deren Hilfe ein Mensch Sprache versteht oder spricht. Wie diese Regeln beschaffen sind und wie sie angewendet werden, ist genetisch festgelegt und wird von ihm als ‚universelle Grammatik' bezeichnet. Somit ist festgelegt, was als menschliche Sprache möglich ist, und was nicht."

Sprache erscheint bei Chomsky also als Menge von Sätzen, die aus einer endlichen Anzahl Elemente auf Grundlage einer endlichen Menge von Regeln gebildet werden kann. Auch die Menge der beiden Sätze {Ich gehe. Ich komme.} gilt nach dieser Definition als Sprache. Sprache heißt also nicht natürliche Sprache. Die Theorie, die den Aufbau einer Sprache beschreibt, erscheint als endliche Anzahl von Regeln, auf deren Grundlage die Sprache hergestellt werden kann – die Herstellbarkeit von Sprache steht von daher außer Frage. Diese Regeln können künstlich – in einem Gerät – oder auch organisch realisiert sein. In der Entwicklung von Programmiersprachen war, wie in diesem Kapitel deutlich wird, eine solches Verständnis von Sprache zentral.

4.2 Wissenschaft und Computereinsatz in den USA der Nachkriegszeit

Um den zeitgeschichtlichen Hintergrund der ersten Programmiersprachenentwicklungen in den USA grob zu umreißen, gehe ich in diesem Abschnitt auf den Stellen-

---

88 Morpheme sind die kleinsten bedeutungstragenden Einheiten in einem Satz.

89 Daß Chomsky auch die organische Umsetzung einer Grammatik mitdachte, verdeutlicht folgendes Gedankenspiel, das er 1956 für „finite state grammars", die allerdings nur einen kleinen Ausschnitt der englischen Sprache erzeugen können, anstellt (S. 20): „In producing a sentence, the speaker begins in the initial state, produces the first word of the sentence, thereby switching into a second state which limits the choice of the second word, etc."

wert von Wissenschaft und Technik in den USA der Nachkriegszeit ein und stelle kurz die Entwicklung des Computereinsatzes in dieser Zeit dar.

Zentral für das Verständnis der gesamten US-amerikanischen Technik- und Wissenschaftsentwicklung der 50er und 60er Jahre ist der durch den zweiten Weltkrieg verursachte Boom der Wirtschaft und der wissenschaftlichen Forschung in den USA.[90] Vor dem Krieg hatte sich die USA in einer ökonomischen Krise befunden; die Wissenschaft fristete ein eher stiefmütterliches Dasein. Die kriegsbedingte Aufrüstung jedoch schaffte eine enorme staatlich finanzierte Nachfrage, die Arbeitslosigkeit in Arbeitskräftemangel und eine stagnierende Produktion in das Bedürfnis nach ständig wachsender Produktivität verwandelte. Ganze Industriezweige wie die Elektronik- und Luftfahrtindustrie entstanden binnen Monaten. Der Umsatz der mit der Herstellung von z. B. Elektronik – Funkgeräten, Radar, Flugzeug- und Steuerungselektronik – befaßten Industrie stieg um 2000%, die Zahl der hier Beschäftigten verfünffachte sich. 1943 war der Bundeshaushalt der USA so groß wie das gesamte Bruttosozialprodukt der vorangegangenen 10 Jahre. Der Krieg war so für die USA ein Wendepunkt, der auch den Alltag der meisten Menschen grundlegend veränderte. Joseph Weizenbaum, später MIT-Professor für Computer Science, erinnert sich entsprechend (Weizenbaum 1999, S. 40):

> „Bei uns gibt es ‚vor dem Krieg' und ‚nach dem Krieg'. [...] an diesem Tag [des Kriegseintritts der USA] sind bei uns alle zur Armee gegangen und für die meisten von uns ist das wirklich ein Meilenstein, so wie hier die Stunde Null oder 1989."

Vor dem Krieg war US-amerikanische Forschung von militärischen und ökonomischen Bedürfnissen noch weitgehend unabhängig. Allgemein wurden Wissenschaftler als unpraktisch veranlagt und für die gesellschaftliche Entwicklung eher unwichtige Gruppe angesehen. So erinnert sich Philip Morse, Physiker und einer der Väter des „Operations Research" (Morse 1977, S. 153f.):

> „Physicists, mathematicians and astronomers were a tiny minority in the country, thought of as impractical, when thought of at all, and most of us were quite satisfied with our insulation from national affairs. Still we wondered what we would be doing if war came."

Der Einsatz hochtechnisierter Waffen im zweiten Weltkrieg machte jedoch den Stellenwert wissenschaftlicher Forschung für den militärischen Bereich deutlich. Wissenschaftler wurden nun an zentraler Stelle in die Rüstungsproduktion eingebunden. Wissenschaftliche Forschung wurde zu einem industriell organisierten Unternehmen, für das finanzielle Beschränkungen keine Rolle spielten (Noble 1986a, S. 10f.): „[...] typically the young physicists at the MIT radiation laboratory had grown accustomed to signing an order for a new instrument whose cost would have deadlocked a faculty

---

90 Im Forschungsprojekt „Sozialgeschichte der Informatik" hat Florian Theißing diese Entwicklung im Rahmen einer Studienarbeit zur „Systemanalyse als politische Planungswissenschaft. Gesellschaftliche Rahmenbedingungen des Einsatzes in den USA der Nachkriegszeit" aufgearbeitet, auf die ich mich im folgenden wesentlich beziehe.

before the war." Es entstand die Kaste der Wissenschaftsmanager, die an die Leitung großer Projekte gewöhnt waren, für die die gesellschaftliche und ökonomische Relevanz ihrer Forschung außer Frage stand und die Forschungsergebnisse marktgerecht zu plazieren versuchten. Die Zusammenarbeit von Forschung, Militär und Industrie wurde zum Alltag – ein Umstand, der bis in die 60er Jahre hinein, und damit in dem in diesem Kapitel besprochenen Zeitraum, in der US-amerikanischen Gesellschaft nicht problematisiert wurde.

Verständlich wird diese unkritische Haltung auf Grund der Rolle, die der zweite Weltkrieg für die USA spielte: Der Zweite Weltkrieg wurde in den USA allgemein als ein gerechter Krieg empfunden, der die Herrschaft des Faschismus über Europa verhindern sollte (Weizenbaum 1999, S. 43):

> „Wenn man als junger Mann auf der Straße in Zivil ging, wurde man gefragt: ‚Warum bist du nicht in der Armee?' Das war fast eine Schande, man mußte und natürlich wollte man auch an dem Krieg gegen Hitler teilhaben."

Zudem brachte der Krieg den USA einen wirtschaftlichen Aufschwung und breiten Wohlstand in bis dahin nicht gekanntem Ausmaß. Die enge Allianz Militär-Industrie-Wissenschaft, die zum US-amerikanischen Kriegserfolg erheblich beigetragen hatte, genoß ein hohes Ansehen. Wissenschaft galt als Garant für technischen Fortschritt und wirtschaftliches Wohlergehen – eine Anschauung, die bis in die 60er Jahre hinein, bis hin zum Vietnamkrieg, im wesentlichen unkritisiert bestehen blieb. Wissenschaftliche Forschung galt als zentrales Mittel der Produktionssteigerung und der militärischen Stärke. Eisenhower, später Präsident und zu diesem Zeitpunkt noch Generalstabschef, forderte 1946 entsprechend (nach Lindner et al. 1988, S. 108):

> „Für die zukünftige Sicherheit der Nation müssen alle zivilen Ressourcen, die im Notfall unsere Hauptunterstützung darstellen, auch in Friedenszeiten eng mit der Armee verbunden werden. [...] Die Streitkräfte hätten den Krieg nicht allein gewinnen können. Wissenschaftler und Wissenschaftsführer stellten uns Techniken und Waffen zur Verfügung, mit denen wir den Feind überlisten und schlagen konnten. [...] Diese Zusammenarbeit muß in eine für Friedenszeiten geeignete Form umgewandelt werden, was nicht nur der Armee den wissenschaftlichen und industriellen Fortschritt nahebringen, sondern auch alle zivilen Ressourcen, die zur Landesverteidigung beitragen können, in die Planung der nationalen Sicherheit einfügen wird."

Eine starke militärische Hochrüstung wurde als notwendig dafür angesehen, die weltpolitische Vormachtstellung erhalten und ausbauen zu können. Das Verteidigungsministerium wurde in der Folgezeit – nun im Angesicht der Frontstellung zu der UdSSR – der mit Abstand größte Finanzier der US-amerikanischen Forschung und Wissenschaft (Leslie 1993, S. 1):

> „In the decade after the Second World War, the Department of Defence (DOD) became the biggest patron of American science, predominantly the physical sciences and engineering but important in many of the natural and social sciences as well. [...] spending for defense research and development surpassed its wartime peak (already fifty times higher than prewar levels) by the end of the Korean War [1953], then climbed to dizzling heights after Sputnik

[1957], reaching $5.5 billion a year by 1969. On average, the DOD accounted for about 80 percent of the federal R&D [Research and Development] budget throughout the 1950s."

Die Entwicklung der ersten Computer in den USA und damit auch die Entstehung der Programmierung fand in den USA im Rahmen der Assoziation aus Militär, Industrie und Wissenschaft statt. Schon der in Kapitel 2 beschriebene Bau des ENIAC, des EDVAC und auch der IAS-Maschine wäre ohne finanzielle Förderung des Militärs nicht denkbar gewesen. Allgemein wurden die ersten Rechenmaschinen auch in der Nachkriegszeit in großzügig von Staat und Militär finanzierten Forschungsprojekten gebaut: Bis Ende der sechziger Jahre wurden alle wichtigen Computerentwicklungen aufgrund von Anforderungen des Militärs spezifiziert und mit Geldern aus dem Verteidigungshaushalt durchgeführt.

Bei den ersten Rechenmaschinen handelte es sich um sehr teure, handwerklich gefertigte Einzelexemplare, für die der Markt und Rentabilitätskriterien noch keine Rolle spielten. Eingesetzt wurden sie für komplizierte wissenschaftliche Berechnungen – ein Bereich, der außerhalb von Militär und Wissenschaft eine zu vernachlässigende Rolle spielte. Der Bedarf an Rechenmaschinen wurde – nicht zuletzt auch auf Grund der hohen Herstellungskosten – entsprechend als sehr gering eingeschätzt. Geschaffen wurde in dieser Zeit jedoch das „Know-how", das die Grundlage für die spätere, auch kommerzielle Herstellung von Rechenmaschinen legte.

In den 50er Jahren war es üblich, die neuen Rechenmaschinen als Elektronengehirne, als „electronic brains" zu betrachten, ähnlich wie das in den in Kapitel 2 und 3 besprochenen Texten der Fall war. „By now the term ‚giant brains' had captured the public imagination", hält z. B. McCorduck (1979, S. 64) für die USA um 1950 fest. Die prinzipielle Gleichartigkeit und Mischbarkeit von Technik und Biologie schien außer Frage zu stehen. Auf dem Titelblatt der Zeitschrift TIME vom 23. 1. 1950 ist z. B. ein „kybernetisches Wesen", wie es dort heißt, abgebildet. Dieses besitzt einen Körper, der wie die damaligen Rechenmaschinen aussieht, verfügt jedoch auch über zwei Arme mit menschlichen Händen und ein menschliches Auge. Die Zeitschrift begrüßte Wiener „als einen der führenden ‚Computermänner', die die Grenzen zwischen den Wissenschaften vom Gehirn, von den psychischen Eigenschaften und der Maschine abschafften" (Galison 1997, S. 312). 1954 sah sich Peter Naur, später Professor für „Datalogie" in Kopenhagen, genötigt, einen Artikel „The Electronic Computer and the Brain" zur Aufklärung der Frage, inwieweit Rechenmaschine und Gehirn vergleichbar sind, zu schreiben (Naur 1954, S. 480):

„Since 1946 a number of complicated electrical devices, known as electronic computers, have seen the light of the day. They have attracted considerable attention, not only among people who normally might be expected to be interested in computing machines, but even among logicians and philosophers. This attention is manifest in the way the devices have been described in newspapers and popular scientific books under remarkable designations, such as electronic brains."

Auch in Deutschland war in der öffentlichen Wahrnehmung die Rechenmaschine zu einem Riesengehirn avanciert, wie eine im Rahmen der Forschungsprojektes „Sozialgeschichte der Informatik" verfaßte Studienarbeit zum Bild der Rechenmaschine im Nachrichtenmagazin „Der Spiegel" zeigt (Balles 1999). Von 1947 – 1970 erschienen im „Spiegel" 158 Artikel, in denen der Begriff „Elektronengehirn" verwendet wurde. Besonders in den Artikeln der 50er Jahre wurden technische und organische Begriffe zu neuen Bezeichnungen für die Rechenmaschine kombiniert: „Elektronengehirne", „Robotergehirne" oder „Maschinengehirne" beflügelten die Phantasie der Leserinnen und Leser. Auch rein organische Bezeichnungen wie „Nervenzellen", „Gehirnzellen" oder „Gehirnhälften" wurden neben Superlativen wie „Supergehirn" oder „Sprachgenie" zur Bezeichnung der Rechenmaschine und ihrer Teile verwendet. In einer Zwischenüberschrift des Artikels „Die Revolution der Roboter" (1955) fand auch der Gedanke der Selbstreproduktion von Automaten seinen populären Ausdruck: „Elektronengehirne vermehren sich selbst" hieß es hier. Weitere klingende Titel wie „Maschinengehirn – Beängstigend menschlich" (1950), „Elektronengehirne – Für intellektuelle Arbeit" (1952) oder „Elektronengehirne. Die Magie der Roboter" (1956) (mit den Zwischenüberschriften „Photozellen statt Augen" oder „Maschinen können lernen") sorgten dafür, daß Rechenmaschine und Gehirn im gesellschaftlichen Bewußtsein untrennbar verknüpft wurden. Die Sicht auf die Rechenmaschine als Gehirn- oder Nervensystemäquivalent, die von Neumann in seinen Texten entfaltet hatte, war in den USA und Europa zu einem Allgemeinplatz geworden.

Im wirtschaftlichen Bereich setzte sich der Computer erst in den 60er Jahren durch.[91] Was die Rechenmaschine für den Einsatz in der Wissenschaft so attraktiv machte – ihre hohe Rechengeschwindigkeit – kam in der Wirtschaft zunächst nicht zum Tragen. Zum einen fehlten für die Haltung massenhafter Daten geeignete Speichermedien, zum anderen leistungsfähige Ein-/Ausgabeeinheiten, denen z. B. auch von Neumann nur wenig Aufmerksamkeit gewidmet hatte (Theißing 1995, S. 16):

> „Der UNIVAC 1 beispielsweise war zwar in der Lage, die Rechenarbeit von Hunderten von Büroangestellten zu ersetzen. Der erste Drucker der UNIVAC schaffte aber mit einer Kapazität von 10 Zeichen pro Sekunde gerade mal die Leistung von zwei Sekretärinnen."

Die Wirtschaft reagierte entsprechend zurückhaltend auf die neuen Maschinen. In der zweiten Hälfte der 50er und auch den 60er Jahren wurde der Einsatz von Computern in Unternehmen immer wieder in Frage gestellt (Theißing 1995, S. 22):

> „In vielen Quellen wird kritisiert, daß Computer oft nur als eine Art Statussymbol angeschafft wurden. Unternehmen würden Computer kaufen, ohne daß ein fundierter Plan vorläge, wofür

---

91 Florian Theißing (1995) hat im Forschungsprojekt „Sozialgeschichte der Informatik" den Einsatz von Computern in US-amerikanischen Unternehmen in den 50er und 60er Jahren aufgearbeitet, was ich im folgenden grob zusammenfasse.

er eigentlich benutzt werden sollte. Diese Entscheidungen wären geprägt von der Angst, ohne Computer gegenüber der Konkurrenz ins Hintertreffen zu geraten oder vom technischen Fortschritt abgehängt zu werden."

Das Bild, das vom Computer als gigantisches und leistungsfähiges Elektronengehirn in den Medien gezeichnet wurde, beeinflußte also auch die Entscheidungen von Unternehmensleitungen. Häufig war allein die damit verbundene Publicity ein wichtiges Motiv für Unternehmensleitungen, eine Rechenmaschine anzuschaffen; Einsparungseffekte hingegen waren nur ungenau zu bestimmen oder überhaupt nicht festzustellen (vgl. Laubach, Thomson 1955, S. 125f.; Theißing 1995, S. 34ff.).

Neben der Presse stilisierten auch die Hersteller von Rechenmaschinen, die auf Grund der zögerlichen Haltung vieler wirtschaftlicher Unternehmen bei der Anschaffung der neuen Maschinen unter Argumenationsdruck standen, diese zu einem unverzichtbaren Bestandteil des modernen Managements (Theißing 1995, S. 23): „So werden in vielen Artikeln des *Harvard Business Review* Unternehmensmanager davor gewarnt, den übersteigerten Verlautbarungen von Herstellern, aber auch WissenschaftlerInnen, allzu viel Glauben zu schenken". Der Historiker Cortada beschreibt die Situation entsprechend wie folgt (Cortada 1987, S. 27):

„The confidence which would be so much in evidence a decade later was nowhere in sight. Vendors had to teach customers why they should use equipment and how to operate it, while trying to justify costs [...]. Major suppliers provided software, education, maintenance, and encouragement. Those tactics continued throughout the 1950s and 1960s."

Ein eigenständiger Softwaremarkt war in den 50er Jahren und bis 1969, dem „unbundeling announcement" von IBM,[92] so gut wie nicht existent. Programme der Rechenmaschinenhersteller wurden als eine Art Service zusammen mit dem Computer verkauft. Die Erstellung von Anwendungsprogrammen wurde häufig von dem jeweiligen Rechnerhersteller oder aber von den Anwendern selbst geleistet. In der Praxis war das Kodieren noch nicht von der Hardware zu trennen, auch wenn von Neumann und Goldstine das Kodieren als ein eigenständiges Wissensgebiet konzipiert hatten. Das Kodieren wurde jedoch, wie es schon in von Neumanns und Goldstines Texten deutlich geworden war, zunächst als eine relativ einfache Tätigkeit angesehen, die leicht zu lernen war (Theißing 1995, S. 43):

„In den ersten Erfahrungsberichten über kommerzielle Computeranwendungen wurde die Programmierung also als eine relativ anspruchslose Tätigkeit angesehen, die wenig Wissen erforderte. Die Fähigkeit zur Produktion von Programmen stellte aus diesem Blickwinkel eher eine Zusatzqualifikation als einen eigenständigen Beruf dar."

---

92 1969 beschuldigte die US-Regierung IBM, das damals führende Unternehmen in der Computerbranche, durch die gemeinsame Vermarktung von Programmen und Rechnern einen fairen Wettbewerb im Bereich der Softwareherstellung zu verhindern. IBM reagierte auf das eingeleitete Anti-Trust-Verfahren mit dem „unbundeling announcement", das den künftig getrennten Verkauf von Rechnern und Programmen zusicherte.

Für das Verständnis des Berufsbilds des Programmierers in den USA war das SAGE-Projekt von großer Bedeutung.[93] Die Airforce begann 1951 in Zusammenarbeit mit der RAND-Corporation, einem im Krieg entstandenen privaten Forschungsinstitut, und dem MIT die Konstruktion eines computergestützten Luftabwehrsystems. Dieses sogenannte SAGE-Projekt war das größte Computerprojekt seiner Zeit, zu dessen Programmierung ca. 600 Personen benötigt wurden, was ungefähr der Gesamtzahl der Programmierer in den USA entsprach. Nötig wurde deshalb die Einrichtung von Ausbildungsplätzen; es wurde eine eigene Organisation zur Rekrutierung und Schulung von Programmierern eingerichtet, die „Systems Development Corporation" (SDC). Im Rahmen sechsmonatiger Crashkurse bildete die SDC bis Ende der fünfziger Jahre 2000 Leute aus – zu dieser Zeit ca. die Hälfte aller US-amerikanischen Programmiererinnen und Programmierer. Die Trennung der Erstellung der mathematischen Problemlösung, die von Neumann und Goldstine als unabhängig von der jeweiligen Rechenmaschine eingeführt hatten,[94] und der Kodierung wurde hier institutionalisiert (Kraft 1977, S. 38):

> „In effect, the SAGE/SDC program prompted the institutionalization on a large-scale of the separation between the conception of a computer program and the physical task of constructing it. More to the point, it created the organizational divisions between the two, including the institutions designed to train the one or the other."

Als Aufgabe des Kodierens galt also, eine bereits gegebene Konzeption eines Programms – was zumeist hieß eine numerische Problemlösung – so zu notieren, daß die Maschine sie ausführen konnte. Obwohl das Kodieren selbst als relativ einfache Tätigkeit galt, zeigen die vielen Klagen über Kommunikations- und Verständigungsprobleme zwischen DV-Personal und Anwendern, daß bei der Erstellung einer Computerlösung oft größere Probleme zu überwinden waren. So heißt es z. B. in einer der ersten Studien über den Rechnereinsatz in Versicherungsunternehmen (Davis 1953, S. 11):

> „However, when an actuary tried to find out how to apply such equipment he quickly learned that life insurance people and electronic engineers were two groups of people who did not speak each other's language. He found the engineers quite willing and even anxious to have their ability and experience applied to the changes necessary to take computers out of purely laboratory work and into the business world, but they were lacking an adequate picture of the facilities the business world needs. On the other hand, actuaries and others in the life insurance field also lacked a sufficient understanding of the equipment which might be reasonably be expected for insurance use. Some medium was necessary to bridge the gap between the two."

Auch in der Folgezeit rissen die Klagen über eine Kluft zwischen „Elektroingenieuren" (heute würde man hier wohl von Informatikern bzw. „computer scientists" reden) und

---

93 Die Geschichte des SAGE-Projektes und besonders auch dessen Verwobenheit mit dem Bau des Whirlwind-Computers, dem größten Digitalcomputerprojekt der späten 40er und frühen 50er Jahre, ist ausführlich in (Redmond, Smith 1980) dargestellt.

94 Vgl. dazu Abschnitt 2.4.2.

Anwendern nicht ab.[95] Typischerweise wurde dabei das in dem Zitat beschriebene Verständigungsproblem und die Unkenntnis des Bereichs des jeweils anderen als das Problem gesehen; um es zu lösen, wurde auf ein neues Medium, eine meist noch ausstehende technische Neuerung verwiesen. Von heute aus gesehen fällt auf, daß die prinzipielle Möglichkeit, eine gewünschte Problemlösung auf dem Computer ausführen zu können, nur selten in Zweifel gezogen wurde. Ein Grund dafür mag sein, daß die Rechenmaschine von Anfang an eng mit dem Bild des „Elektronengehirns" verwoben war. Dieser Sinnzusammenhang legt nahe, mehr oder weniger selbstverständlich davon auszugehen, daß die Rechenmaschine sich wie ein besonders leistungsfähiger Mensch in einen Arbeitsablauf eingliedert. Anzunehmen ist, daß sich aus den Besonderheiten des Einsatzes einer Maschine – anstelle eines oder mehrerer Menschen – ergebende Probleme häufig in anderer Gestalt verhandelt wurden: so z. B. als die oben geschilderte Verständigungsproblematik, als Klagen über einseitig technisch ausgerichtete Programmierer oder deren unstrukturiertes Vorgehen.

4.3 Kodiersysteme

Im Bereich der Kodierung wurde zunächst angestrebt, die Arbeit des Kodierens zu erleichtern. Eine Möglichkeit bot hier zunächst das Maschinendesign: Eine Rechenmaschine sollte ein Instruktionenset bereitstellen, das eine möglichst komfortable Bedienung gewährleistete (Hopper, Mauchly 1953, S. 1250):

> „The design of the general-purpose computer must always be a compromise involving many different requirements [...]. One of the factors which has always seemed to deserve heavy weight in shaping such decisions has been the ease or difficulty with which the human programmer can adapt himself to the instruction code peculiar to his own computer."

Neben dem Hardware-Design wurden zudem Programmiertechniken zur Vereinfachung des Kodierens entwickelt: Es wurden, wie schon im „First Draft of a Report on the EDVAC" für Diskussionen über den Code vorgeschlagen, „short symbols", mnemonische Codes[96], statt Binärzahlen zur Bezeichnung der Instruktionen eingeführt. Als ein nächster Schritt wurden Speicheradressen und Registernummern in Dezimalstatt in Binärdarstellung ausgedrückt. 1951 schlugen Wilkes, Wheeler und Gil die Verwendung von „subroutines" vor, die in Bibliotheken zusammengefaßt werden und solche Codeteile beinhalten sollten, die in verschiedenen Problemstellungen immer wieder verwendet wurden. Zudem führten sie die sogenannte „relative Adressierung" ein, bei der der Code in mehrere Blöcke aufgeteilt wird und in jedem Teil alle Adressen relativ zu einer Konstante angegeben werden. Der Code konnte so geändert

---

95 Genauer wird das Beklagen solcher Probleme in (Theißing 1995, S. 53ff.) geschildert.

96 Man spricht von mnemonischen Codes, wenn die kodierte Darstellung eines Wortes das kodierte Wort erkennen läßt (z. B. SUB als Bezeichnung für die Subtraktion).

werden, ohne daß alle Adressen in dem gesamten Programm neu bestimmt werden mußten. In einem nächsten Schritt wurden Befehle wie Adressen durchgehend symbolisch notiert, so daß bei Änderungen auch innerhalb einzelner Blöcke die Adressen nicht mehr neu von der kodierenden Person berechnet werden mußten.

Auf Wilkes, Wheeler und Gill (1951) geht auch der Begriff „Assembler" (dtsch. Monteur) zurück. Als Assembler wird das Programm bezeichnet, das die im Code angeführten Unterroutinen aus der Bibliothek in das Gesamtprogramm „montiert", aus den relativen bzw. symbolischen die tatsächlichen Adressen berechnet und den mnemonischen Code in die interne Maschinendarstellung übersetzt. Ein besonders verbreiteter Assembler war SAP (Symbolic Assembly Program), mit dessen Hilfe eine ganze Generation von Programmierenden die Benutzung der IBM 705 erlernte.[97]

Von Sprachen war im Bereich der Kodierung bis Mitte der 50er Jahre nur vereinzelt die Rede. Die ersten, heute als Vorläufer höherer Programmiersprachen betrachteten Systeme hießen z. B.

- „Short Code" (vorgeschlagen von Mauchly 1949 für die UNIVAC),
- „AUTOCODE" (frühe Nutzung 1952 für die Manchester Mark I),
- „Speedcoding" (entwickelt ab 1952 für die IBM 701 unter der Leitung von John Backus),
- „Laning and Zierler System" (Entwicklungsbeginn 1952 am MIT für den Whirlwind Computer),
- „A-0, A-1, A-2, A-3" (entwickelt ab 1951 bei Remington Rand unter der Leitung von Grace Hopper für die UNIVAC),[98]
- „BACIAC" (Boeing Aircraft Company Algebraic Interpretive Coding System, 1955 für die IBM 701).

Im Falle der meisten hier genannten Systeme wurde im wesentlichen auf die eigentliche Maschine eine Oberfläche gesetzt, die automatische Adressierung und Fließkommaverarbeitung[99], verbesserte Ein-/Ausgabevorrichtungen und einfachere Nota-

---

97 Mehr über diese frühen Versuche, die Programmierung zu vereinfachen, findet sich in (Wegner 1976), (Knuth, Trabb Pardo 1980) und in (Sammet 1969). Jean Sammet war selbst entscheidend an der Entwicklung von COBOL beteiligt und schrieb 1969 ein sehr ausführliches Buch über die Geschichte von Programmiersprachen, in dem sie systematisch die je beteiligten Organisationen, Personen und die wichtigen technischen Features detailliert darstellt.

98 Bei der Beschreibung des A-0 tauchte zum ersten Mal der Begriff „compile" im Bereich des „automatic programming" auf. 1954 beschrieb Moser den Begriff so, wie er im A-2-Compiler umgesetzt wurde (Moser nach Knuth, Trabb Pardo 1980, S. 234): „To compile means to compose out of materials from other documents. Therefore, the compiler method of automatic programming consists of assembling and organising a program from a program or routines or in general from sequences of computer code which have been made up previously."

99 Bei automatischer Fließkommaverarbeitung muß beim Kodieren nicht mehr für jede verarbeitete Zahl die Anzahl der Stellen vor bzw. nach dem Komma angegeben und damit im Voraus berechnet und eingeschätzt werden.

tionsweisen bot. Eine mathematische Schreibweise erlaubten die Systeme in der Regel jedoch nicht (Sammet 1969, S. 5):

> „However, most of these systems had a fixed format and, in particular, did not permit the writing of mathematical expressions in anything resembling natural notation. Only the Laning and Zierler system and BACIAC had this latter facility."

Eine geschriebene Programmzeile wurde zumeist interpretiert und direkt ausgeführt. Nur die A-Serie kompilierte das gesamte Programm, wobei eine Programmzeile normalerweise direkt äquivalent zu mehreren Zeilen Maschinencode war. Eine mit einem solchen System ausgerüstete Rechenmaschine wurde zumeist als eine neue Pseudo-Maschine gedacht, wie das am Beispiel des Whirlwind-Computers in folgendem Zitat deutlich wird. Auf Grund dessen, daß drei verschiedene Interpreter für diese Maschine zur Verfügung standen, wurde über drei verschiedene Maschinen gesprochen (Gorn 1959, S. 117):

> „The Whirlwind Computer at MIT was made available to a wide class of users [...] in the form of three pseudo-computers. These three machines existed only as programmed ‚interpreters', known as ‚The Summer Session Computer', ‚The Comprehensive Computer', and Laning and Zierler's ‚Algebraic Computer'."

Obwohl die Rechenmaschine zur Erleichterung des Kodierens also durch Programme und nicht mehr auf der Hardware-Ebene verändert wurde, wurde das Resultat der Programmierung doch als eine neue Maschine vorgestellt.

Der herausstechendste Nachteil dieser frühen Systeme war, daß sie einen Maschinencode produzierten, der eine sehr viel höhere Laufzeit hatte, als ein direkt von einem Kodierer geschriebenes Assembler- oder Maschinenprogramm. Die interpretativen Systeme verlangsamten die Maschine um den Faktor fünf bis zehn. Die erzielten Vereinfachungen beim Schreiben des Programms waren so gering, daß sie solche Laufzeiterhöhungen nicht rechtfertigten. In der Praxis waren diese Systeme deshalb nur wenig verbreitet; als einen Grund dafür nennt rückblickend Backus, daß von Protagonisten des sogenannten „automatic programming" – wie das in der Presse, aber auch von Seiten der Wissenschaft in bezug auf Rechenmaschinen nicht ungewöhnlich war – enorm hohe, technisch letztlich nicht erfüllbare Erwartungen geweckt wurden (Backus 1981, S. 26):

> „Another reason that ‚automatic programming' was not taken seriously by the computing community came from the energetic public relations efforts of some visionaries to spread the world that their ‚automatic programming' systems had almost human abilities to understand the language and needs of the user;"

In den 50er Jahren wurde es üblich, Bemühungen, den Prozeß des Kodierens zu beschleunigen, unter dem – auch von Backus in dem Zitat benutzten – Stichwort „automatic programming" oder auch als „automatic coding" zu diskutieren. Eines der ersten Treffen zum Thema „automatic programming" fand im Mai 1954, gesponsert vom „Office of Naval Research" statt. Das nächste wichtige Treffen war das Symposium „Advanced Programming Methods for Digital Computers", gesponsort

vom „Navy Mathematical Computing Advisory Pannel“ und dem „Office of Naval Research“ im Juni 1956 (vgl. Sammet 1969, S. 4f.). Beim „automatic programming“ oder „automatic coding“ ging es nicht mehr so sehr darum, die Arbeit des Kodierens zu vereinfachen, als vielmehr darum, den Kodierer zu ersetzen. Die Maschine sollte ihre Programme nun möglichst weitgehend selbst erstellen. Nach dem „American Standard Vocabulary for Information Processing“ von 1966 der damaligen „American Standards Association“ meinte „automatic programming“ (nach Sammet 1969, S. 13): „The process of using a computer to perform some stages of the work involved in preparing a program.“ „Automatic programming“ oder auch „automatic coding“ sollten also, wie es diese Ausdrücke schon besagten, Teile des Programmierprozesses automatisieren – oder, wie es Grace Hopper 1962 ausdrückte, dazu dienen, den Kodierer loszuwerden (Hopper nach Greenberger 1962, S. 272): „Work in automatic coding has concentrated on getting rid of the coder.“ Ähnlich formulierte der Schweizer Heinz Rutishauser 1956 sein Verständnis des „automatischen Programmierens“ (nach Eulenhöfer 1998b, S. 121):

> „Darunter verstehe ich Methoden, welche das Programmieren durch eine geeignete Aufschreibung der dem Problem zugrundeliegenden Formeln völlig zu mechanisieren gestatten.“

Die Entwicklung von FORTRAN, auf die ich im nächsten Abschnitt eingehe, war Teil der Bemühungen um das „automatic coding“. Das FORTRAN-Projekt trat mit dem Ziel an, den professionellen Kodierer dadurch zu ersetzen, daß die Maschine sich selbst kodierte.

4.4 Das „FORmula TRANslating System“

Das „FORmula TRANslating System“ wurde ab 1954 bei IBM in New York unter der Leitung von John Backus entwickelt. Ich werde im folgenden zunächst die (Vor)Geschichte der FORTRAN-Entwicklung[100] und John Backus‘ Rolle näher beschreiben. Deutlich wird dabei, daß erst mit der IBM 704 die technischen Voraussetzungen geschaffen waren, auf deren Grundlage das FORTRAN-Sytem erstellt werden konnte – besonders die Fließkommaverarbeitung war hier zum ersten Mal hardwaretechnisch realisiert. Auf Initiative von Backus wurde für die IBM 704 das FORTRAN-Projekt gestartet, das zu den größten Programmierprojekten der Zeit gehörte. Mit dem Vorhaben, den Kodierer maschinell zu ersetzen, stieß es zunächst auf Skepsis und Unverständnis.

---

100 Backus selbst hat auf einer Konferenz zur Geschichte der Programmiersprachen 1978 seine Sicht auf die Geschichte von FORTRAN ausführlich dargelegt (Backus 1981).

Im Februar 1957 stellte die FORTRAN Gruppe zum ersten Mal das gerade fertiggestellte FORTRAN System einer größeren Fachöffentlichkeit vor. In den Proceedings der „Western Joint Computer Conference" von 1957 findet sich das zu dem Vortrag gehörige Papier, als dessen Autorinnen bzw. Autoren alle Mitglieder des Projektes, gruppiert nach Institutionen und in alphabetischer Reihenfolge, aufgeführt sind (Backus et al. 1957). Das Papier beschreibt – nebst einer Einleitung und einem Schlußabschnitt, die vor allem der Darstellung der Motivationen und der Bewertung des Systems gewidmet sind – die beiden Teile des FORTRAN Systems: zunächst die „FORTRAN language" und danach den „translator", der Programme in der „FORTRAN language" in „machine language"-Programme übersetzt. In Abschnitt 4.4.2 gehe ich ausführlich auf dieses Papier und besonders das Bild ein, das es von der FORTRAN-Sprache und dem „translator" zeichnet. In 4.4.3 fasse ich meine Ausführungen zu FORTRAN zusammen.

4.4.1 Vorgeschichte

John Backus wurde am 3. Dezember 1924 in Philadelphia geboren.[101] Im Alter von 18 Jahren begann er ein Chemiestudium, das er schon nach einem Semester abbrechen mußte, da er wegen häufigen Fehlens aufgefallen war. Im Frühjahr 1943 wurde er zur Armee einberufen. Dort war er zunächst in einem Flugabwehrprogramm in Georgia beschäftigt. Von September 1943 bis März 1944 besuchte er ein spezielles Ingenieurtraining an der Universität von Pittsburgh und wurde im Anschluß zu einem sechsmonatigen medizinischen Armeekurs im Krankenhaus von Atlantic City gesendet. Im März 1945, nachdem er sich einer komplizierten Operation wegen eines Knochentumors am Kopf unterzogen hatte, begann er ein Medizinstudium, das er ebenfalls wieder abbrach. Nach einer weiteren Operation wurde er nach seiner Genesung aus der Armee ausgemustert und entschloß sich, eine Ausbildung als Fernseh- und Radiotechniker zu beginnen. Einer der Leiter dieser Ausbildung weckte sein Interesse für Mathematik; er begann zunächst nebenbei Kurse an der Universität von Columbia zu belegen und entschied sich dann ganz für ein Mathematikstudium, das er 1950 mit dem „Master's Degree" abschloß. In dieser Zeit hörte er von einem Bekannten von dem IBM-Rechner SSEC (Selective Sequence Electronic Calculator), der in der 57. Straße in New York, dem damaligen Hauptsitz von IBM, stand.

Die SSEC war die erste elektronische Rechenmaschine von IBM. IBM hatte sich bis dahin auf dem elektronischen Rechenmaschinenmarkt sehr zurückgehalten und auch ein Angebot zur Zusammenarbeit von Eckert und Mauchly ausgeschlagen, nachdem diese die Moore-School verlassen hatten (Lee 1982, S. 2):

---

101 Die Bemerkungen zur Biographie von Backus stützen sich auf (Slater 1987).

„IBM was a comparative latecomer in the electronic computer market, T. J. Watson Sr. having little confidence in the reliability of machines which one could not actually see working, and perhaps influenced in some degree by the prognosis that a small number of such machines would satisfy the world's demands for computation."

Als Backus die SSEC besichtigte, wurde er von einer IBM-Angestellten herumgeführt, die ihn auch zu einem Vorstellungsgespräch bei IBM motivierte. 1950 wurde Backus bei IBM für die Programmierung der SSEC eingestellt. Er hatte zuvor mit Computern so gut wie gar nichts zu tun gehabt. Die Faszination, die von diesem Rechner, und wohl auch Rechnern allgemein in dieser Zeit ausging, wird sehr schön deutlich in folgender Interviewpassage, in der sich Backus an seine Arbeit mit der SSEC erinnert:[102]

„And it was fascinating to see, you know, the SSEC occupied a room larger than the whole top for this house. [...] On one side lots of big relays made a huge amount of noise. On the other side a lot of paper tape reading stations made a lot of noise, too. And lights, you know, flushing. During the time I worked there they made a science fiction movie there. It was an ideal setting for that. Huge console lights and switches – you could really see things happening. Anyway, working on the SSEC was great fun. At all the time you had two or three or four engineers sitting there, helping to fix all the problems that arose."

1952 begann Backus, auf der IBM 701 zu arbeiten, die er in dem Interview im Gegensatz zur SSEC als „bunch of grey refrigerators with boxes with nothing to see or hear" beschrieb. Er leitete ein kleines Programmierteam, das Speedcoding entwickelte. Dieses System setzte auf die IBM 701 eine Oberfläche, die Fließkommaverarbeitung, Indexregister und verbesserte Ein-/Ausgabe anbot. Speedcoding wurde 1953 fertiggestellt. Wie später bei FORTRAN stand bei der Entwicklung von Speedcoding im Vordergrund, Kosten dadurch zu sparen, daß die Zeit des Programmierens und damit Personalkosten reduziert wurden. So schrieb Backus in einem kurzen Artikel, der Speedcoding vorstellte (Backus 1954, S. 4):

„It is clear, therefore, that programming and testing cost comprise between 50 and 75% of the total cost of operating a computing installation. Since Speedcoding reduces coding and testing time considerable, and is fairly fast in operation, it will often be the more economical way of solving a problem."

Kunden sollten mit Speedcoding eine höhere Rechenzeit eintauschen gegen geringere Personalkosten – und damit den Gewinn von IBM, die Rechnerzeit in Servicecentern vermietete oder in Gestalt von Rechenmaschinen verkaufte, steigern. Backus nahm auch Teil an der Entwicklung der IBM 704 und drängte immer wieder darauf, für diese Maschine Indexierung und Fließkommaverarbeitung hardwaretechnisch zu realisieren, was zunächst – besonders für die Fließkommaverarbeitung – aus Angst um Profit beim Rechnerzeitverkauf auf heftigen Widerstand stieß (Lee 1982, S. 3):

„Backus championed this cause against strong opposition which was based on the profit motives of not giving away too much of the computational pie; after all, such niceties would

---

102 Dieses Interview führte ich mit Backus am 6. 9. 1994 in San Francisco.

only decrease the time needed to complete a computation in the service bureau thus decreasing the profit to be gained."

Die IBM 704 wurde entgegen dieser Bedenken nach einigen firmeninternen Auseinandersetzungen die erste Maschine, in der Indexierung und Fließkommaverarbeitung hardwaretechnisch realisiert waren. Systeme wie Speedcoding wurden damit in weiten Teilen überflüssig.

Im Dezember 1953 verfaßte Backus einen Brief an seinen Vorgesetzten Cuthbert Hurd, in dem er die Entwicklung eines „automatic programming system" für die IBM 704 vorschlug, das zum einen mathematiknahe Notation erlauben und zum anderen effizienten Maschinencode erzeugen sollte; auf Grund der hardwaretechnisch realisierten Fließkommaverarbeitung schien das nun technisch machbar. Hurd, der große Stücke auf Backus setzte – wie in einem Interview deutlich wurde, das ich mit ihm im September 1996 führte – bewilligte diesen Vorschlag. Erwartet wurde, daß das Projekt binnen sechs Monaten abgeschlossen sein würde. Obwohl das FORTRAN-Projekt in der Folgezeit wieder und wieder verlängert werden mußte, zahlte IBM für die immer weiter wachsenden Personal- und Ausstattungskosten. Es dauerte drei Jahre, bis im April 1957 die Arbeit an dem Übersetzungsprogramm (engl. translator) abgeschlossen war.[103] Zu dieser Zeit umfaßte die FORTRAN-Gruppe 13 Personen.

Anfang 1954 wurde zunächst Irving Ziller, im Mai Harlan Herrick (bevor er 1949 zu IBM kam, Mathematiklehrer)[104] und Robert A. Nelson in dem FORTRAN-Projekt angestellt. In einem Mitte Mai auf dem oben erwähnten „Naval Research Symposium on Automatic Programming" eingereichten Papier von Backus und Herrick ist die Frage formuliert, die die FORTRAN Gruppe in ihrer Arbeit leitete (nach Backus 1981, S. 28): „ [...] can a machine translate a sufficiently rich mathematical language into a sufficiently economical program at a sufficiently low cost to make the whole affair feasible?" Daß zu der damaligen Zeit dieser Frage mit sehr viel Skepsis begegnet wurde, wird deutlich in dem „Preliminary Report, Specifications for the IBM Mathematical FORmula TRANslating System, FORTRAN", der im November 1954 fertiggestellt wurde. Hier sahen sich die Autoren genötigt, über mehrere Seiten hinweg ihr Vorhaben zu rechtfertigen, „to enable the IBM 704 to accept a concise formulation of a problem in terms of a mathematical notation and to produce automatically a high-speed 704 program for the solution of the problem." (nach Sammet 1969, S. 143) Die in dem „Preliminary Report" gegebene Sprachbeschreibung entsprach übrigens bereits bis auf relativ wenige Änderungen der späteren FORTRAN 1 Version. Die noch

---

103 Der Hergang der Arbeit, die Verteilung der Aufgaben und die verschiedenen Schwierigkeiten, mit denen die Gruppe konfrontiert wurde, ist in (Backus 1981) sehr ausführlich dargestellt. Ich werde mich deshalb hier auf eine sehr geraffte Darstellung beschränken.

104 Soweit mir Näheres über die berufliche Biographie der beteiligten Personen bekannt ist, erwähne ich das im Anschluß an ihre erste Nennung.

vorgenommenen Änderungen sind fast alle dem Wunsch nach effizienter Objektcodeerstellung geschuldet.

Nach Erscheinen des „Preliminary Report" hielten Backus, Herrick und Ziller fünf bis sechs Vorträge bei Kunden, die eine IBM 704 bestellt hatten und trugen die Ideen zu FORTRAN vor. Neben der Information der Kunden sollten diese Vorträge auch dazu dienen, Einwände oder besondere Kundenanforderungen an das neue System zu sammeln. Die Rückmeldungen waren jedoch sehr gering. Die Zuhörer reagierten vor allem skeptisch. Besonders die Behauptung, daß der „translator" effizienten Maschinencode erzeugen sollte, wurde nicht geglaubt. Nur ein Vortrag bei der „United Aircraft Corporation" Anfang 1955 führte zu einem Erfolg: Es wurde vereinbart, daß Roy Nutt, ein Experte für Ein-/Ausgabe und Assemblerroutinen, das FORTRAN-Projekt unterstützen sollte.[105]

Anfang der 50er Jahre arbeitete John von Neumann 30 Tage pro Jahr als Berater bei IBM. Er assistierte bei der Entwicklung der IBM 704 und stand für die Beratung bei mathematischen Problemen bereit. Hurd organisierte 1954 eine FORTRAN-Präsentation, um von Neumanns Meinung zu dem Projekt einzuholen. Neben Hurd waren hier Backus und Frank Beckmann – IBM-Manager für „Pure Programming" und Überlieferer dieses Treffens (Beckmann 1980, S. 177f.) – anwesend. Von Neumann reagierte auf FORTRAN zwar nicht direkt negativ, war aber von dem Vorhaben ganz und gar nicht überzeugt (Beckmann nach Lee 1982, S. 12):

> „I do not know if von Neumann expressed any opinion about FORTRAN outside this meeting, but I would certainly not describe his reaction as being unduly negative – somewhat apathetic perhaps, but not strongly negative. [...] In general von Neumann was not enthusiastic of automated programming aids."

Von dieser Einstellung war denn auch Neumanns Antwort auf die Präsentation von FORTRAN geprägt. Er fragte (ebd.): „Why would you want another language?" Bereits in den „Planing and Coding Problems" hatten Goldstine und von Neumann wiederholt ihre Überzeugung ausgedrückt, daß das Kodieren – besonders wenn mit Hilfe von Flußdiagrammen durchgeführt – eine einfache Tätigkeit sei (vgl. Abschnitt 2.4.2). Die Notwendigkeit für FORTRAN war für von Neumann entsprechend nicht ersichtlich.

Die Mitglieder der FORTRAN-Gruppe, die nach 1954 immer weiter anwuchs, hatten teilweise mit Computern vor ihrer Arbeit an FORTRAN noch keinerlei Erfahrung und kamen nach einem einführenden Kurs zur Programmierung der IBM 704 in die Gruppe. Neben Herrick und Backus waren auch Sheldon Best, Lois M. Haibt und Richard Goldberg ausgebildete Mathematiker. Mehrfach wurden Leute für ganz andere

---

105 Neben Nutt spielte noch ein weiterer „Leiharbeiter", Sheldon Best, in der FORTRAN-Entwicklung eine wichtige Rolle. Ende 1955 bis 1956 wurde er mit Einverständnis seines Arbeitgebers, dem „Digital Computer Laboratory" des MIT, für die Programmierung des „translators" eingesetzt.

Aufgaben – wie z. B. „technical typing", d. h. Sekretariatsarbeit, (1954 Nelson) oder das Verfassen von Handbüchern (1956 David Sayre, Physiker im Gebiet der Kristallographie) – angestellt, waren dann aber auch an der Programmierung des „translators" beteiligt. Einmal wurde auch ein Schachmeister als Programmierer angestellt, weil man dachte, daß dieser gut programmieren können müßte, was sich jedoch als falsch herausstellte. Die gesamte Einstellungspolitik war typisch für die Art und Weise, wie in dieser Zeit Programmierer und Programmiererinnen rekrutiert wurden. So erinnert sich Lois M. Haibt:[106]

> „At that time it was very hard to find programmers. There were all sorts of theories like: go out and find chess-players because they ought to be good at programming; and anybody who liked to play chess was hired. At one time we had a philosopher, a geologist, a chess grand master and so on. So it was a much more open field because they were looking for anybody they could find."

Die Programmierung war also ein neues, offenes und aufregendes Gebiet. Die FORTRAN-Gruppe arbeitete zudem in dem Bewußtsein, entgegen der Skepsis der Außenwelt etwas ganz Neues zu versuchen. Die Arbeitsatmosphäre wird immer wieder als besonders gut, aufregend und familiär beschrieben (Interview Backus):

> „One thing they all agree, that everyone agrees, is that it was the most enjoyable project they ever worked on. We all got along well together and we had this marvellous support. We were trying to do something we didn't know how to do when we started. And during the whole time IBM just kept providing resources and never we had to figure out budget or anything. We just had the resources we needed. We didn't have all this nasty burocratic stuff that goes on today."

Die großzügige Unterstützung der Firma und das Bewußtsein, eine Pionierarbeit zu leisten, die zudem allgemein für unleistbar gehalten wurde, schufen eine Arbeitsumgebung, die zusammenschweißte, in der man sich wohlfühlte, hochmotiviert zusammenarbeitete und sich gegenseitig unterstützte (Interview Haibt und Goldberg, Sprecherin Haibt):

> „It was very much a cooperative effort. If somebody would find out some silly mistake that they made it was a big joke at lunch time. If one person said: How do you think that works? the two people maybe went to the third person and then the three people went to the fourth one and so on."

Wie Haibt und Goldberg in dem Interview beschrieb auch David Sayre die Arbeit in dem FORTRAN-Projekt als einen Höhepunkt in seinem beruflichen Leben. Auch in seiner Erzählung wird deutlich, daß die Gruppe in seiner Sicht etwas ganz Neues leistete, was sie zu einer Art Familie zusammenschweißte:[107]

> „As far as sociology is concerned it was in many respects the happiest period of my scientific life. It was a good family feeling. The thing that sticks in my mind most of all were two things; one was lunch time and the other was night time work. [...] The last year FORTRAN-

---

106 Ausschnitt aus einem Interview, das ich am 15. 9. 1994 mit Lois M. Haibt und Richard Goldberg im IBM Research Center in Yorktown Heights, New York, führte.

107 Das Interview mit David Sayre führte ich am 14. 9 1994 in seinem Haus in St. James, New York.

people worked almost as much during night as they did during the day. At night we could get long periods of machine time. There is something special about working at night. Here we were in the middle of Manhattan, the streets were deserted and these eight or ten people would come and work through the night and leave as the sun was rising in the morning. And from time to time Ann, my wife, would come in at about midnight with a picnic bascet of food. I think everybody enjoyed it and felt that we were doing something that nobody had ever done before."

Als 1957 der „translator" schließlich zu laufen begann, waren die Resultate für die Mitglieder der Gruppe selbst erstaunlich – ein Umstand, der auch in ihrem Papier „The FORTRAN Automatic Coding System" deutlich wird und auf den ich weiter unten noch genauer eingehe (Backus 1981, S. 36):

„[...] when later on we began to get fragments of compiled programs out of the system, we were often astonished at the surprising transformations in the indexing operations and in the arrangement of the computation which the compiler made, changes which made the object program efficient but which we would not have thought to make as programmers ourselves." (Backus Interview): „When the compiler first started producing results, it was really very exciting. It was a pretty complex process, so you couldn't really know in advance what it was going to produce. It was sometimes puzzling to see how the job described by the FORTRAN program really represented this machine program. That was really fascinating. Like observing a natural phenomenon."

4.4.2 Text: „The FORTRAN Automatic Coding System"

Im Februar 1957 präsentierte die FORTRAN-Gruppe das fertiggestellte FORTRAN-System auf der „Western Joint Computer Conference" einer größeren Fachöffentlichkeit. Während ich bisher die Geschichte des FORTRAN-Projektes vor allem auf Grundlage der Erinnerungen der Beteiligten dargestellt habe, gehe ich nun genauer auf dieses Zeitdokument ein. Ich arbeite heraus, daß der Text und seine Begrifflichkeit durchzogen sind von der Grundidee, mit dem FORTRAN-System den Kodierer zu ersetzen: Angesprochen werden die Vorgesetzten des Kodierers, die Arbeit des Kodierers erscheint im Vergleich mit der Maschine als langsam und fehlerhaft; die FORTRAN-Sprache scheint der mathematischen Sprache weitgehend zu entsprechen, der „translator" die Arbeit des Kodierers – das Übersetzen der mathematischen Problembeschreibung in den Code – durchzuführen. Wie der Kodierer erscheint der „translator" als selbständig handelnd, wobei er einerseits als Maschine und andererseits als Programm – als Symbolsystem – gedacht wird. Ähnlich wie in den von Neumannschen Texten dokumentiert sich so die Hybridobjekt-Sichtweise, in der die Grenzen zwischen Mensch, Maschine und Symbolsystem verschwimmen.

4.4.2.1 Adressaten und Selbstpräsentation

Das Papier „The FORTRAN Automatic Coding System“ richtet sich weniger an den tätigen Kodierer, der – so könnte man es jedenfalls darstellen – durch FORTRAN von vielen alltäglichen Mühen befreit wird. Vielmehr spricht es technisch geschulte potentielle Käufer und Wissenschaftler an, wie sie auf der Konferenz zugegen waren: Die Steigerung der Effizienz und damit verbundene Kostenersparnisse erscheinen als Hauptargumente für den Erfolg des FORTRAN-Systems. Es wird versucht, die Vorteile des Systems zahlenmäßig zu fassen: als Geld-, Zeit- und Arbeitsersparnis oder als Verringerung der Ausbildungszeit. Auch die „conciseness“ (dtsch. Präzision, Kürze, Exaktheit) der FORTRAN-Sprache und ihre „convenience“ (dtsch. Annehmlichkeit, Zweckmäßigkeit) wird gemessen: erstere als Verhältnis von Maschineninstruktionen zu „FORTRAN-statements“ (4 bis 20 zu 1), letztere als das Verhältnis von Kodierzeit in Assemblersprache zu Programmierzeit in FORTRAN-Sprache (ebenfalls 4 bis 20 zu 1) (vgl. Abschnitt Conciseness and Convenience, S. 42f.)[108]. Daß das Papier weniger für den Kodierer als vielmehr für dessen Vorgesetzten geschrieben ist, wird auch daran deutlich, daß nicht auf die leichte Erlernbarkeit der FORTRAN-Sprache abgehoben wird. Statt dessen wird betont, daß es leicht ist, sie zu lehren (Abschnitt Education, S. 44): „It is considerably easier to teach people untrained in the use of computers how to write programs in FORTRAN language than it is to teach them machine language.“

Die 13 Autoren (12 Männer und eine Frau) selbst kommen in dem Papier, das in sehr unpersönlichem Stil gehalten ist, fast gar nicht vor. Zwar werden vor der Beschreibung der einzelnen Teile des „translator“-Programms die jeweiligen Entwickler gesondert aufgeführt; das – wie auch der Umstand, daß die gesamte Gruppe als Autor fungiert – zeigt, daß explizit die von allen einzelnen eingebrachte Leistung gewürdigt und nicht als Gesamtes dem Leiter, Backus, oder der IBM zugeschrieben werden soll. Geschrieben ist das Papier jedoch aus der Perspektive eines anonymen Beobachters, der das FORTRAN-Projekt, die Sprache und den „translator“ beschreibt – anders als das z. B. in den besprochenen Texten von Neumanns der Fall ist, der als „Ich“ oder „Wir“ schrieb. Die in dem Papier gemachten Aussagen erscheinen dadurch, daß sie auf einen scheinbar außenstehenden Beobachter zurückgehen, als objektiver. Diese durch den Stil nahegelegte Objektivität des Standpunktes wird auch auf andere Weise untermauert. So wird herausgestellt, daß die Erfahrungen der FORTRAN-Gruppe mit

---

108 Da ich das Papier nur als Kopie der fertigen Version, die die FORTRAN-Gruppe einreichte, nicht jedoch in der in dem Tagungsband abgedruckten Version erhalten konnte, stimmen meine Seitenzahlen nicht mit denen im Tagungsband überein. Ich gebe deshalb immer den Abschnitt mit an, in dem das jeweilige Zitat zu finden ist.

dem von ihr selbst entwickelten FORTRAN-System nur eine bedingte Aussagekraft in bezug auf die Nützlichkeit des Systems haben (vgl. Introduction, S. 2, Conclusions, S. 45). Die Autoren erscheinen so als ehrlich gegenüber den Zuhörenden. Zudem präsentieren sie sich als vor allem intellektuell an dem FORTRAN-System interessiert. Der Vortrag schließt mit der Feststellung:

> „In any case the intellectual satisfation of having formulated and solved some difficult problems of translation and the knowledge and experience acquired in the process are themselves almost a sufficient reward for the long effort expended on the FORTRAN project."

Während die Zuhörer in dem Papier als diejenigen erscheinen, die FORTRAN auf Grund erwarteter Kostenersparnisse schätzen, scheint die intellektuelle Befriedigung – ein ideeller Wert – für die Autoren bzw. IBM schon genug Belohnung zu sein. Unterstrichen wird so, daß es den Autoren nicht darum geht, um jeden Preis zu verkaufen. Das und die scheinbare Objektivität ihres Standpunktes erlauben, Vorteile des Systems darzustellen, ohne daß man – zumal Kostenersparnisse in den entsprechenden Argumentationen eine große Rolle spielten – an eine Verkaufsveranstaltung erinnert wird.

4.4.2.2 Der Kodierer und die Maschine

In dem letzten Abschnitt wurde deutlich, daß sich der Text vor allem an die Vorgesetzten des Kodierers, an potentielle Käufer und Wissenschaftler, wendet. Der Kodierer kommt entsprechend in dem Papier schlecht weg: Im Gegensatz zu der Maschine wird von ihm ein Bild gezeichnet, nach dem er langsam ist und Fehler macht. Das zeige ich im folgenden kurz auf.

Zu Beginn des Papiers wird die Aufgabe des FORTRAN-Projekts wie folgt charakterisiert:

> „The FORTRAN project was begun in the summer of 1954. Its purpose was to reduce by a large factor the task of preparing scientific problems for IBM's next large computer, the 704. If it were possible for the 704 to code problems for itself and produce as good programs as human coders (but without errors), it was clear that large benefits could be achieved."

Ziel des FORTRAN-Projekt ist also, den Aufwand der Aufbereitung wissenschaftlicher Probleme für die IBM 704 entscheidend zu reduzieren. Um dieses Ziel zu erreichen wird angestrebt, daß die IBM 704 Probleme für sich selbst kodieren und dabei, allerdings ohne Fehler zu machen,[109] so gute Programme produzieren soll wie ein

---

109 Hier wird zwar der Eindruck erweckt, nur der Kodierer mache Fehler; die Rechenmaschinen der Zeit (z. B. die SSEC, s. o.) – und auch das FORTRAN-System selbst, wie in dem Text (Abschnitt Debugging, S. 45) zugegeben wird – waren jedoch ebenfalls relativ fehleranfällig. Vermutlich wird davon ausgegangen, daß die Maschine zumindest prinzipiell fehlerfrei sein kann, während das vom Menschen nicht zu erwarten ist.

menschlicher Kodierer.[110] Die IBM 704 wird also als Konkurrent des menschlichen Kodierers gezeichnet und soll dessen Arbeit nun selbst übernehmen. „Klar" ist, daß dadurch großer Nutzen gewonnen wird, weil die Arbeit eines menschlichen Kodierers als langsam, fehlerhaft und damit teuer im Gegensatz zur Maschine erscheint: Große Vorteile – so die Autoren – könnten qua Selbstkodierung durch die IBM 704 aus folgendem Grund erreicht werden (4. und 5. Satz):

> „For it was known that about 2/3 of the cost of solving scientific and engineering problems on large computers was that of problem preparation. Furthermore, more than 90% of the elapsed time for a problem was usually devoted to planning, writing and debugging the program."

Würde die Maschine sich selbst kodieren, würden also Kosten und Zeitverbrauch beim Planen und Schreiben des Programms sowie der Fehlerbeseitigung – und damit in den Phasen, in denen der menschliche Kodierer involviert ist – entscheidend gesenkt: Ohne menschlichen Kodierer kaum noch Kosten – so jedenfalls der hier nahegelegte Schluß. Nur ein Drittel der Kosten und weniger als 10% des Zeitverbrauchs scheinen – trotz damals sehr teurer Rechnerzeit – der Maschine zuzuschlagen zu sein.

Durchaus möglich wäre, die Vorteile des FORTRAN-Systems anders herauszuarbeiten als auf dem Wege dieser einseitigen Abwertung der menschlichen Kodierarbeit. Z. B. könnte das FORTRAN-System als Verbesserung der noch mangelhaften Bedienungsmöglichkeiten der Maschine dargestellt werden. Im Gegensatz zum Kodierer und Programmierer wird in dem Papier jedoch ein Bild der Rechenmaschine gezeichnet, das diese als groß, schnell, fehlerlos und aktiv erscheinen läßt. Die IBM 704, für die das FORTRAN-System entwickelt wurde, wird als „large computer" eingeführt, der sich eigentlich auch selbst kodieren kann. Während die Maschine als makellos erscheint, wirkt der Kodierer langsam und fehlerhaft. Den menschlichen Kodierer durch weitere Technik soweit möglich zu ersetzen, erscheint als die folgerichtige Konsequenz. Während zuvor Kodiersysteme, die ebenfalls die Phase der Problemvorbereitung, die Kodierung, verkürzen und damit kostengünstiger gestalten sollten, als Unterstützung des Kodierers durch z. B. mnemonischen Code oder Programmbibliotheken dargestellt wurden,[111] trat das FORTRAN-Projekt mit dem Ziel an, den Kodierer durch die Maschine zu ersetzen.

---

110 Bereits im „Preliminary Report" von 1954 wurde das Ziel ähnlich formuliert (Backus 1981, S. 30): „On the basis more of faith than of knowledge, we said that ‚FORTRAN may apply complex, lengthy techniques in coding a problem which the human coder would have neither the time nor the inclination to derive or apply.' "

111 1951 schrieben Wilkes, Wheeler und Gill z. B. zur Einführung von Subroutinenbibliotheken (S. 15): „It is highly desirable that the machine itself should carry out as much of this work as possible [gemeint waren routinemäßige Arbeiten wie das Berechnen einer Wurzel]; the chance of error is then reduced and the programmer is left free to concentrate his attention on the more essential aspects of the program." Zwar wird hier auch die Maschine als Instanz gezeichnet, die Teile der Arbeit des Programmierers übernehmen und dabei Fehler vermeiden kann; die Bibliotheken werden jedoch als Erleichterung für den Programmierer dargestellt. Seine Arbeit wird dadurch aufgewertet, daß er sich den eigentlich wichtigen Aufgaben widmen kann.

4.4.2.3 Das FORTRAN-System

Das FORTRAN-System war exklusiv für die IBM 704 hergestellt; daran, daß es auch auf andere Rechner übertragen werden könnte, wurde zunächst nicht gedacht (Backus 1981, S. 30): „We certainly had no idea that languages almost identical to the one we were working on would be used for more than one IBM computer, not to mention those of other manufacturers." Das FORTRAN-System bestand aus zwei Teilen: der „FORTRAN language" und dem „translator", der „FORTRAN language"-Programme in Maschinenprogramme übersetzen sollte (Backus et al. 1957, Introduction, S. 2):

> „It [the FORTRAN system] has two components: the FORTRAN language, in which programs are written, and the translator or executive routine for the 704 which effects the translation of FORTRAN language programs into 704 programs."

Bei der Programmierung mit FORTRAN mußte das Programm zunächst in FORTRAN-Sprache geschrieben und anschließend auf Lochkarten gestanzt werden. Zum Übersetzen des Programms mußte im Anschluß das „translator program" in die IBM 704 eingelesen werden. Der „translator" war dann das einzige Programm, das auf der Maschine lief. Die Bedienung der „translator"-Maschine wird in dem FORTRAN-Papier von 1957 entsprechend in einer Weise beschrieben, die an konventionelle Maschinenbedienung erinnert; der Kartenstapel mit dem FORTRAN-Programm kann nun, so der Text, in den Kartenleser gelegt werden (Abschnitt Translation, S. 16): „The deck of cards obtained by keypunching may then be put in the card reader of a 704 equipped with the translator program." Wird dann der „load button" gedrückt, so heißt es im folgenden, gebe die Maschine im Falle der fehlerfreien Übersetzung einen Stapel „binary cards" oder ein Magnetband, ein „binary tape", mit dem entsprechenden 704-Programm aus. Dieses konnte dann seinerseits in die IBM 704 eingelesen und ausgeführt werden.

Die ursprüngliche Kodierarbeit wurde durch das FORTRAN-System in zwei Teile gesplittet: das Schreiben der Programme in „FORTRAN language" wurde weiterhin vom Menschen erledigt, während ein Teil der ehemaligen Kodierarbeit an den „translator" delegiert und dadurch automatisiert wurde. Interessant ist deshalb die Frage danach, welcher Teil der ehemaligen Kodierarbeit beim Programmierer gelassen und welcher Teil von der FORTRAN-Gruppe als automatisierbar konstruiert und durch den „translator" realisiert wurde. Ich werde, um diese Frage zu beantworten, zunächst die Konzeption der Sprache und anschließend die des „translators" näher betrachten. Dabei wird sich zeigen, daß in dem Papier das „Programmieren" als Tätigkeit gezeichnet wird, die der Arbeit des Ingenieurs oder Wissenschaftlers, der eine mathematische Problemlösung erstellte, weitgehend entspricht. Unterschiede zwischen der FORTRAN-Sprache und der mathematischen Sprache erscheinen als marginal. Das Kodieren hingegen wird weitgehend als eine Art Buch- oder Haushaltertätigkeit dargestellt. Da zu dieser Zeit in US-amerikanischen Unternehmen Tätig-

keiten einfacher Angestellter durch Rechenmaschineneinsatz automatisiert wurden, schloß diese Sichtweise an einen in wirtschaftlichen Unternehmen erfolgsversprechend scheinenden Ansatz des Computereinsatzes an.

4.4.2.3.1 Die FORTRAN-Sprache

In dem Papier wird begrifflich unterschieden zwischen dem „coder", der in Maschinensprache „kodiert", und dem „programmer", der in „FORTRAN language" „programmiert". Während das Kodieren, wie oben aufgezeigt, als langsam und fehleranfällig erscheint, geht das Programmieren den Autoren zufolge leicht und schnell von der Hand (Abschnitt Conciseness and Convenience, S. 42):

> „The statement of a program in FORTRAN language rather than in machine language or assembly program language is intended to result in a considerable reduction in the amount of thinking, bookkeeping, writing and time required."

Zum einen vermeidet die Benutzung der FORTRAN-Sprache, so der Text, Fehler (Conclusions, Debugging, S. 44): „Furthermore, the nature of the language makes it possible to write programs with far fewer errors than are to be expected in machine-language programs." Zum anderen scheint die Sprache leicht zu lernen zu sein: Nicht-Programmierer können sie, so heißt es in den „Conclusions" im Abschnitt „Education", in ein- bis dreitägigen Kursen erlernen. Um diese leichte Lernbarkeit noch zu unterstützen, wurde gleichzeitig mit der ersten Version ein sehr gut strukturiertes, mit vielen Beispielen anschaulich arbeitendes Handbuch herausgegeben, für dessen Erstellung eine zusätzliche Person, David Sayre, in dem FORTRAN-Projekt eingestellt wurde. In dem Abschnitt „Education" wird auch auf dieses Handbuch, das übrigens bereits vor dem FORTRAN-System selbst fertiggestellt war und, so der Text, speziell als „teaching tool" designed ist, verwiesen.[112]

Die Einfachheit und Schnelligkeit des Programmierens leitet sich in dem Text vor allem aus Ähnlichkeit der FORTRAN-Sprache zu der mathematischen Sprache ab. So soll die FORTRAN-Sprache erlauben, daß der Programmierer in einer exakten, eindeutigen und quasi mathematischen Sprache nur noch den – so wird es zumindest nahegelegt – generellen Plan entwerfen muß; die zeit- und kostenaufwendige Arbeit der Kodierung sollte die Maschine selbst übernehmen. Die Aufgabe des Kodierens und Debuggens (der Fehlersuche) würde sich damit – so die Erwartung der Autoren – auf 1/5 des zuvor benötigten Aufwandes reduzieren (6. - 8. Satz):

---

112 Im FORTRAN-Projekt wurde auf eine gute Dokumentation der Sprache, die insbesondere der leichten Erlernbarkeit dienen sollte, sehr großen Wert gelegt. 1957 wurde von Grace Mitchell ein „Programmer's Primer" erstellt, der wie das Manual mit vielen Beispielprogrammen arbeitete, aber didaktisch noch besser aufbereitet war. Beide – das Manual und besonders auch der Primer – hatten nach Einschätzung von Backus (1980, S. 36ff.) großen Einfluß auf den raschen Erfolg und die schnelle Verbreitung von FORTRAN.

„In many cases the development of a general plan for solving a problem was a small job in comparison to the task of devising and coding machine procedures to carry out the plan. The goal of the FORTRAN project was to enable the programmer to specify a numerical procedure using a concise language like that of mathematics and obtain automatically from this specification an efficient 704 program to carry out the procedure. It was expected that such a system would reduce the coding and debugging task to less than one fifth of the job it had been."

Mit Hilfe der FORTRAN-Sprache scheint also nur noch ein genereller Plan für die Problemlösung entworfen bzw. ein numerisches Verfahren spezifiziert werden zu müssen – eine Aufgabe, die im wissenschaftlichen Bereich 1957 in der Regel einem dem professionellen Kodierer vorgesetzten Naturwissenschaftler, Ingenieur oder Mathematiker oblag, der in der „Sprache der Mathematik" bewandert war. Die Arbeit an der Maschine scheint damit von den Personen, die bis dahin die Kodierung vorbereitet hatten, selbst ausgeführt werden zu können – der professionelle Kodierer wird hingegen überflüssig. In der Terminologie von Neumanns und Goldstines (1947) ausgedrückt scheint das „Programmieren" der Phase der mathematischen Problemvorbereitung des Kodierens zuzurechnen zu sein (oder zumindest sehr nah an diese heranzurücken), die diesen beiden Autoren zufolge noch nichts mit der Maschine zu tun hatte und sich aus der Notwendigkeit des Berechnens an sich ergab (vgl. Abschnitt 2.4.2).

Der Anspruch, daß es sich bei der FORTRAN-Sprache um eine mathematikähnliche Sprache handelt, findet in dem Papier zunächst in der verwendeten Terminologie seinen Ausdruck. So gibt es „arithmetic statements", ein Ausdruck, der auch synonym zu „arithmetic formula" benutzt wird (vgl. Abschnitt Section 2, S. 27; Section 1, S. 22). Auch ein FORTRAN-Programm als ganzes erscheint als eine Art mathematischer Formel, was sich besonders in dem Namen „FORmula TRANslating System" widerspiegelt – übersetzt werden soll eine mathematische Formel in Maschinensprache. In dem Manual von 1956 heißt es diesbezüglich im ersten Satz sehr prägnant (IBM 1956, S. 2): „The IBM Mathematical Formula Translating System FORTRAN is an automatic coding system for the IBM 704".[113]

Die Mathematikähnlichkeit der FORTRAN-Sprache wird in „The FORTRAN Automatic Coding System" nicht argumentativ begründet, sondern qua Anschauung anhand von Beispielen vermittelt. In dem ersten Satz zur Sprachbeschreibung heißt es:

---

113 1954 hatten Backus und Herrick auf den „Symposium on Automatic Programming" die Idee, daß ein Programmierer im wesentlichen nur noch das numerische Verfahren niederschreiben müßte, wie folgt beschrieben (Backus, Herrick nach Knuth, Trabb Pardo 1980, S. 237): „A programmer might not be considered too unreasonable if he were willing only to produce the formulas for the numerical solution of his problem, and perhaps a plan showing how the data was to be moved from one storage hierarchy to another, and then demand that the machine produce the results for his problem. No doubt if he were too insistent next week about this sort of thing he would be subject to psychiatric observation. However, next year he might be taken more seriously."

„The FORTRAN language is most easiliy described by reviewing some examples." Die Bedeutung der einzelnen Beispiele und die Regeln des sprachlichen Aufbaus (in heutiger Terminologie: Semantik und Syntax) werden anhand dieser Beispiele zusammen und nicht-formal beschrieben. Die Darstellung ist sehr anschaulich und gut verstehbar gehalten, was die versprochene Einfachheit der Sprache illustriert. Verschiedenheiten zwischen mathematischer und der FORTRAN-Notation werden dabei nicht thematisiert: so z. B. weder die unterschiedliche Bedeutung des Gleichheitszeichens (s. u.) noch der Umstand, daß in der FORTRAN-Sprache (bedingte) Sprungbefehle existieren. Für von Neumann und Goldstine (1947, vgl. Abschnitt 2.4.2) war das noch ein Grund gewesen festzustellen, daß ein Text nicht einfach aus der „Sprache der Mathematik" in die „Sprache des Codes" übersetzt werden kann. Dieses Argument könnte genauso auf die FORTAN-Sprache angewendet werden: Da auf Grund von Sprungbefehlen die „FORTRAN statements" nicht linear abgearbeitet werden und im Falle bedingter Sprungbefehle die Reihenfolge der Abarbeitung von Zwischenresultaten abhängt, die erst während des Maschinenlauf erzielt werden, muß der Zusammenhang zwischen mathematischem Text und FORTRAN-Programm als ein dynamischer verstanden werden. Der mathematische Text kann demnach nicht statisch in ein FORTRAN-Programm übersetzt werden; kennzeichnend dafür ist, daß bis heute auch bei der Programmierung mit FORTRAN (oder anderen imperativen Programmiersprachen) vor der Erstellung des eigentlichen Programms häufig Flußdiagramme gezeichnet werden.

Daß die Ähnlichkeit der FORTRAN-Sprache zu mathematischen Notationsweisen in dem FORTRAN-Text nicht problematisiert wird, kann damit erklärt werden, daß die FORTRAN-Notation tatsächlich mathematischen Notationsweisen ähnlicher war, als das bis dahin in der Programmierung üblich war. Die FORTRAN-Sprachgestaltung konnte auf Grund des niedrigen Standards in bezug auf die Mathematikähnlichkeit der Kodiersysteme relativ sorglos angegangen werden. Als das eigentliche Problem sahen die FORTRAN-Entwickler die Erstellung eines leistungsfähigen, effizienten Objektcode erzeugenden Übersetzerprogramms. Die Beschreibung der „FORTRAN language" nimmt in dem Papier entsprechend nur die Hälfte der Darstellung des „translators" ein. Schon diese Gewichtung macht deutlich, daß die Gestaltung der Sprache – und damit die Herstellung einer mathematikähnlichen Sprache – im Gegensatz zur Programmierung des „translators" als ein einfaches Problem erschien, das weniger systematisch als eher intuitiv bearbeitet wurde (Backus 1981, S. 30):

> „As far as we were aware, we simply made up the language as we went along. We did not regard language design as a difficult problem, merely a simple prelude to the real problem: designing a compiler which would produce efficient programs. [...] Very early in our work we had in mind the notions of assignment statement, subscripted variables and DO statements [...]. We felt that these provided a good basis for achieving our goals for the language, and whatever else was needed emerged as we tried to build a way of programming on these basic ideas."

Die Selbstverständlichkeit, mit der die FORTRAN-Sprache als mathematikähnlich eingeschätzt wurde, kann auch in Zusammenhang mit der allgemeinen Wahrnehmung der Rechenmaschine gesehen werden. Wie ich in Abschnitt 4.2 aufgezeigt habe, galt die Rechenmaschine in den 50er Jahren in der Öffentlichkeit (nicht nur in der USA) als ein großes Elektronengehirn. Daß dieses Gehirn „rechnet", schien – wie schon in von Neumanns Texten – außer Frage zu stehen. Zu der Annahme, daß es eine mathematische Sprache versteht, ist es nur ein kleiner Schritt. Das Sinngeflecht, in das die neue Maschine eingebettet war, ließ diese Annahme leicht als fraglose Selbstverständlichkeit erscheinen.

So selbstverständlich, wie die FORTRAN-Sprache als mathematikähnlich eingeschätzt wurde, wurde sie auch als „Sprache" gesehen. Das erklärt sich zum einen aus dem zeitgenössischen Sprachverständnis (vgl. Abschnitt 4.1), zum anderen kann auch diese Begrifflichkeit als Ausdruck des engen Sinnzusammenhangs, in dem Rechenmaschinen und Gehirn standen, gesehen werden: Wie ein Gehirn schien auch eine Rechenmaschine eine Sprache verstehen zu können – eine Metaphorik, die bereits in den in Kapitel 2 besprochenen Texten aufschien. Im folgenden stelle ich einige Überlegungen dazu an, wie die Wahrnehmung von FORTRAN als „Sprache" die Sicht auf das Programmieren beeinflußte. Um zunächst ein grobes Bild der ersten Version der FORTRAN-Sprache, die in dem Papier beschrieben ist, zu vermitteln, führe ich einige Beispiele auf, die dem Text entnommen sind. Dabei wird deutlich, daß an die Stelle der Instruktionen bzw. Befehle des Codes in dem FORTRAN-Papier – einhergehend mit der durchgängigen Verwendung des Sprachbegriffs – „statements" (dtsch. Aussage, Erklärung, Behauptung, Darlegung) treten, so z. B.

- „arithmetic statements", Zuweisungen der Form

  ```
  ROOT = ( – ( B/2.0 ) + SQRTF(( B/2.0 )**2 – A*C ))/A
  ```

  Erläuterung (Backus et al. 1957, Abschnitt Arithemtic Statements, S. 4): „Its meaning is: ‚evaluate the expression on the right of the = sign and make this the value of the variable on the left'. " Das Gleichheitszeichen, das in der Mathematik die Gleichheit zweier Instanzen markiert, wird hier zu der Handlungsaufforderung, einen Wert einer Variablen und damit letztlich einer Speicherzelle zuzuweisen. Während also in der FORTAN-Sprache das „statement" ROOT = ROOT + 2 eine korrekte Verwendung des Gleichheitszeichens darstellt, ist eine solche Gleichung in der Mathematik falsch.
- „DO statements", wie z. B.

  ```
  4 DO 5 I = 1, 1000
  ```

Erläuterung (Abschnitt DO Statements, S. 7): „Statement 4 says ‚DO the following sequence of statements down to and including the statement numbered 5 for succesive values of I from 1 to 1000'."

- „IF statements", wie z. B.

IF (ARG – ALPHA(I)) 4, 3, 3

Erläuterung (Abschnitt READ, PRINT, ..., S. 14)): „The IF statement says ‚If ARG – ALPHA(I) is negative go to statement 4, if it is zero go to statement 3 and if it is positive go to 3'."

- „GO TO statements", wie z. B.

GO TO 2

Erläuterung (ebd.): „The GO TO statement, of course, passes control to statement 2".

- „READ statements", wie z. B.

READ 1, ALPHA, RHO, ARG

Erläuterung (ebd., S. 13): „The READ statement says ‚READ cards in the card reader which are arranged according to the FORMAT statement 1 and assign the successive numbers obtained as values of ALPHA(I) I = 1, 25 and RHO(I) I = 1, 25 and ARG'." Im Gegensatz zu z. B. von Neumann oder auch den LISP-Entwicklern legte das FORTRAN-Projekt großen Wert auf die Behandlung der Ein-/Ausgabe, die zu dieser Zeit auf Grund der vielen verschiedenen Ausgabegeräte und -formate kompliziert zu handhaben war (FORTRAN Manual, S. 24):

„There are 13 FORTRAN statements available for specifying the transmission of information, during the running of the object program, between core storage on the one hand and tapes, drums, and attached card reader, card punch, and printer on the other hand."

- das „DIMENSION statement", z. B.

DIMENSION A(1000), B(1000)

Erläuterung (Abschnitt DO Statements, DIMENSION Statements ..., S. 7): „[...] the second statement merely informs the executive routine that the vectors A and B each have 1000 elements."

- „function statements", wie z. B.

ROOTF(A,B,C) = (– (B/2.0) + SQRTF((B/2.0)**2 – A*C))/A

Erläuterung (Abschnitt Function Statements, S. 5): Function statements „are composed of the desired function name (ending in F) followed by any desired arguments which appear in the arithmetic expression on the right of the = sign."

- das „FORMAT statement", wie z. B.

FORMAT(5F12.4)

Erläuterung (Abschnitt READ, PRINT ..., S. 12): „The FORMAT statement says that numbers are to be found (or printed) 5 per card (or line), that each number is in Fixed point form, that each number occupies a field 12 columns wide and that the decimal point is located 4 digits from the right."

Die FORTRAN-Sprache besteht also aus Aussagen, aus „statements". Diese können, wie die Erläuterungen zu den ersten fünf oben angeführten Beispielen zeigen, befehlenden Charakter haben, werden aber nicht als „orders" oder „instructions" bezeichnet. Dem entspricht, daß weder der „translator", von dem die „statements" ja bearbeitet werden, noch seine Sektionen oder Teile – folgt man der in dem Papier gewählten Begrifflichkeit – bloß Anweisungen befolgen oder ausführen. Statt dessen werden z. B. „arithmetic formulas" behandelt (engl. „treat") oder „arithmetic instructions" generiert, die ihrerseits die Berechnung ausführen (engl. „carry out") (vgl. Abschnitt Section 1, S. 22). Nur ganze Aufgaben (engl. „task") werden ausgeführt (vgl. ebd., oder Abschnitt Section 2, S. 27), nicht jedoch einfache Anweisungen. Im Gegensatz dazu wird, wenn es um Maschinensprache geht, in dem Text durchweg von „instructions" oder „machine instructions" gesprochen. Mit der durchgängigen Einführung der Sprachmetapher tritt die Rechenmaschine dem Programmierer also anders entgegen: Während beim Kodieren der Maschine befohlen wird, werden der mit dem „translator" ausgerüsteten Maschine beim Programmieren Mitteilungen gemacht.

Die scheinbaren Hauptakteure in dem Teil des Papiers, das die FORTRAN-Sprache beschreibt, sind die „statements". Sie scheinen zu informieren, zuzuweisen, zu verursachen (engl. „to cause"), zu leiten (engl „to direct"), zu kontrollieren oder etwas zu sagen (s. o. Erläuterung der „statements"). Auch in dem Teil, in dem der „translator" beschrieben wird, erscheinen sie als handelnd (Abschnitt „Case of subscripts not controlled by DO's", S. 32): „[...] a value may be read in by a READ statement, it may be calculated by an arithmetic statement [...]". Nur einmal, bei der Einführung der „arithmetic statements" (s. o.), wird von der Bedeutung, die ein „statement" hat,

gesprochen. Ansonsten wird die Bedeutung, wie in obigen Beispielen, meist dadurch erläutert, daß die „statements" ihren Inhalt selbst sagen.

Neben dieser Fähigkeit zu sprechen, die für eine „Aussage" recht nahe liegt, erscheinen die „statements" besonders auch als diejenigen, denen die Steuerung bzw. Kontrolle obliegt (s. o., z. B. in der Erläuterung zu GO TO) und die die Ausführung anderer „statements" veranlassen. Wenn in der Sprachbeschreibung davon die Rede ist, daß ein oder mehrere „statements" ausgeführt werden, erscheint bis auf einmal ein „DO statement" als die Instanz, die diese Ausführung veranlaßt. Bei den „statements" wird also explizit die initiierende und die Kraft zu handeln verortet.

„Statements", Aussagen oder Erklärungen, wird in dem Text also die Kraft zu handeln, und besonders auch eine Handlung zu initiieren, zugesprochen – ähnlich wie in den von Neumannschen Texten „Instruktionen" die Fähigkeit zur Aktion besitzen zu schienen. In den in Kapitel 2 besprochenen Texten wurden diese Instruktionen einerseits als im Speicher abgelegte technische Artefakte gedacht, und es wurde auch ihre Ähnlichkeit zu maschinellen Bauteilen (und in Kapitel 3 zu Genen) diskutiert. Andererseits erschienen die Instruktionen auch als Symbole oder Zeichenketten mit mathematischer Bedeutung, die z. B. die logische und algebraische Definition eines Problems beschreiben sollten. In dem FORTRAN-Text werden die „statements" nun kaum noch in ihrer technischen Gestalt gedacht: wie sie z. B. im Speicher technisch realisiert sind, ist in diesem Text kein Thema mehr.[114] Ihre Eigenschaft, technisches Artefakt zu sein, tritt damit in den Hintergrund. Sie erscheinen vor allem als mathematikähnliche Aussagen, mit deren Hilfe ein numerisches Verfahren spezifiziert werden kann. Die Fähigkeit zur Operation scheint ihnen jedoch weiterhin innezuwohnen. Bei „statements" handelt es sich damit um agierende Notationen, die im Gegensatz zu den Instruktionen stärker in ihrer Eigenschaft als Symbole oder Zeichenketten und weniger als technische, gegenständliche Artefakte gedacht werden.

Ein weiterer Unterschied zwischen Instruktionen und „statements" besteht darin, daß Instruktionen – und auch Programme gedacht als Folge von Instruktionen – einen Aufruf zur Handlung beinhalten und damit auf eine ausführende Instanz verweisen. Wird ein Programm hingegen als Folge von Aussagen oder Erklärungen vorgestellt, tritt dieser Verweis auf eine ausführende Instanz – und damit der dynamische Charakter eines Programms – in den Hintergrund.[115] Entsprechend hielten von

---

114 Nur einmal erscheinen sie in ihrer materiellen Realisierung auf Lochkarten (Abschnitt Translation, S. 16): „The deck of cards obtained by keypunching may then be put in the card reader of a 704 equipped with the translator program."

115 Bei von Neumann schienen Instruktionen auch zu beschreiben, so z. B. einen zu reproduzierenden Automaten oder eine logische und algebraische Problemdefinition. In dem FORTRAN-Papier haben umgekehrt „statements" – Aussagen, die beschreiben – befehlenden Charakter (z. B. GO TO 2). Beiden – Instruktionen wie „statements" – wird die Kraft zu handeln zugeschrieben. Das Beschreiben eines Sachverhalts, das Anweisen und die Aktion werden in beiden Begriffswelten zusammengedacht.

Neumann und Goldstine noch fest, daß ein Befehl nicht nur für sich selbst, sondern für alle Durchläufe der Steuereinheit durch den Befehl steht – daß Befehl und Maschinenlauf also nicht getrennt voneinander betrachtet werden können (vgl. Abschnitt 2.4.2). Im Gegensatz zu von Neumann und Goldstine thematisiert die FORTRAN-Gruppe diese Differenz zwischen mathematischem Text und Programm nicht mehr. Ein „FORTRAN statement" und auch ein FORTRAN-Programm scheint weitgehend einer mathematischen Formel zu entsprechen. Die Rechenmaschine scheint so für den Prozeß des Programmierens kaum noch eine Rolle zu spielen.

4.4.2.3.2 Der FORTRAN-Übersetzer

Der „translator" bewirkte die Übersetzung von „FORTRAN language programs" in IBM 704-Programme. Im Gegensatz zu den meisten bis dahin üblichen „automatic coding systems" interpretierte er ein FORTRAN-Programm nicht Zeile für Zeile, sondern übersetzte das ganze Programm und nahm dabei – entsprechend dem Anspruch, so guten Code wie ein menschlicher Kodierer zu produzieren (s. o., Abschnitt 4.4.2.2) – weitgehende Optimierungen vor (Backus et al. 1957, Abschnitt Method of Translation, S. 46):

> „In general the translation of a FORTRAN program to a machine language program is characterized by the fact that each piece of the output program has been constructed, instruction by instruction, as not only to produce an efficient piece locally but also to fit efficiently into its context as a result of many considerations of the structure of its neighbouring pieces and of the entire program."

Der „translator" beachtet bei der Erstellung der einzelnen Maschineninstruktionen also nicht nur die Struktur benachbarter Programmteile, sondern auch die des gesamten Programms. Wie ein guter Übersetzer eine natürliche Sprache in eine andere, übersetzt der FORTRAN-Übersetzer aus der Ursprungssprache nicht einfach Wort für Wort oder Satz für Satz, sondern berücksichtigt auch den Kontext, in dem ein übersetzter Satz schließlich steht. Das mag ein Grund dafür sein, daß in dem FORTRAN-Projekt zu dem Begriff „translator" statt dem des „compilers" gegriffen wurde. Wie in dem Papier durchgängig der Sprachbegriff (FORTRAN-Sprache und Maschinensprache) statt z. B. Kodiersystem verwendet wird, wird die Aufgabe des „translators" als Übersetzen einer Sprache in eine andere und nicht etwa als Zusammenstellen oder Zusammentragen (engl. compile) verstanden. Am Ende ihres Vortrags stellen die Autoren entsprechend einen Zusammenhang zwischen ihrer Arbeit und dem Übersetzen allgemein her:

> „In any case the intellectual satisfaction of having formulated and solved some difficult problems of translation and the knowledge and experience acquired in the process are themselves almost a sufficient reward for the long effort expended on the FORTRAN project."

Ihre Arbeit erscheint hier als Formulierung und Lösung einiger schwieriger Probleme der Übersetzung. Angedeutet wird, daß der „translator", der ihre Problemlösungen ja realisiert, auch einen Beitrag zu Fragen der Sprachübersetzung allgemein leisten kann.

Der „translator" wird in dem Papier als „executive routine for the 704 which effects the translation" (Introduction, S. 2) eingeführt. Als „executive routine" wird er in seiner Eigenschaft als Programm gesehen, das die Übersetzung bewirkt. Zudem erscheint er auch als Maschine. So heißt es (Abschnitt Preparation of a Program for Translation, S. 16): „The translator accepts statements punched one per card"; der „translator" akzeptiert also Aussagen oder Erklärungen, wenn sie auf eine Lochkarte gestanzt sind – wird hier also als materielle, mit Lochkarten zu versorgende Instanz gesehen. Vier Sätze später ist von der IBM 704, die mit dem „translator program" ausgerüstet ist, die Rede: „The deck of cards obtained by keypunching may then be put in the card reader of a 704 equipped with the translator program." Als „translator", der die auf Lockarten gestanzten „statements" akzeptiert, erscheint hier die mit dem Übersetzerprogramm ausgerüstete IBM 704. Die Perspektive auf den „translator" wechselt also: er erscheint mal als Programm, mal als Maschine.

Der „translator" erscheint in dem Papier als handelnd: Folgt man den Formulierungen der Autoren produziert er das 704-Programm (vgl. Abschnitt READ, PRINT ..., S. 14), akzeptiert (vgl. Abschnitt Preparation of a Program for Translation, S. 16), benötigt (vgl. Abschnitt General Organization of the System, S. 20), findet Fehler und zeigt sie an (und die er nicht findet, sind angeblich vom Programmierer ganz leicht zu finden, vgl. Abschnitt Debugging, S. 44) und stellt Überlegungen an (Abschnitt Method of Translation, S. 46). Auch die einzelnen Teile, die „sections", aus denen der „translator" besteht, lesen, klassifizieren, stellen und setzen zusammen, mischen, führen Analysen durch und konvertieren (vgl. Abschnitt General Organization of the System, S. 17f.), treffen Entscheidungen oder machen die Arbeit (vgl. Abschnitt Section 4 and Section 5, S. 38).

Neben solchen den „translator" als handelnde Entität zeichnenden Formulierungen wird der „translator" auch als quasi selbständig dargestellt. So wird in dem Text anhand der von dem „translator" erstellten Programme die „Natur" des Übersetzungsprozesses „beobachtet", fast so als würden die Autoren den „translator" gar nicht kennen (Abschnitt Method of Translation, S. 47):

> „There are a number of interesting observations concerning FORTRAN-written programs which may throw some light on the nature of the translation process. Many object programs, for example, contain a large number of instructions which are not attributable to any particular statement in the original FORTRAN program. Even transfers of control will appear which do not correspond to any control statement (e.g. DO, IF, GO TO) in the original program. The instructions arising from an arithmetic expression are optimally arranged, often in a surprisingly different sequence than the expression would lead one to expect. Depending on

its context, the same DO statement may give rise to no instructions or to several complicated groups of instructions located at different points in the program."

Die Entwickler selbst stellen hier ihr Produkt vor, als würde es Überraschendes und quasi Unvorhersehbares leisten, das die Gruppe nach dem Abschluß der Arbeit selbst nur noch beobachten kann. Die erzeugte Instruktionen sind – wie es in dem Zitat herausgestellt wird – nicht einfach auf ein bestimmtes „FORTRAN-statement" zurückzuführen, sondern in Abhängigkeit vom Kontext optimal und ganz anderes, als man das erwarten würde, arrangiert. Auch in Interviews, die ich mit mehreren Mitgliedern der Gruppe führte, wurde dieses Staunen über die vom „translator" erzeugten Maschinenprogramme noch Jahrzehnte später thematisiert (vgl. Abschnitt 4.4.1). Das hat sicherlich damit zu tun, daß der „translator" und die von ihm vorgenommenen Optimierungen arbeitsteilig programmiert wurden; die einzelnen konnten von daher nicht die Funktionsweise des gesamten Programms vollständig überblicken. Daß der „translator" im Text als etwas quasi Eigenständiges erscheint, muß jedoch ebenso in Zusammenhang mit dem Anspruch des FORTRAN-Projektes gesehen werden, ein System zu entwickeln, das zumindest ebensogut, wenn nicht besser als ein Mensch kodieren (oder auch eine Sprache in eine andere übersetzen) kann. Gesehen durch die Folie des menschlichen Kodierers werden dem „translator" solche Eigenschaften zugesprochen, die eigentlich als menschliche Privilegien erscheinen: die Fähigkeiten, selbständig zu produzieren, zu entscheiden, zu überlegen sowie Arbeit zu erledigen und dabei den Kontext zu berücksichtigen.

Damit der „translator" mit dem menschlichen Kodierer konkurrieren kann, ist, so der Text, der größte Teil des für den „translator" programmierten Codes der Optimierung des zu erstellenden IBM 704-Programms gewidmet (Abschitt General Organization of the System, S. 20):

„A final observation should be made about the organization of the system. Basically, it is simple, and most of the complexities which it does possess arise form the effort to cause it to produce object programs which can compete in efficiency with hand-written programs."

Besonderer Wert wird dabei auf die optimale Behandlung sogenannter „housekeeping" oder auch „bookkeeping operations" gelegt. Damit sind alle Operationen gemeint, die nicht direkt mit der Durchführung einer Berechnung zusammenhängen, wie sie in einem „arithmetic statement" beschrieben ist: die Verwaltung des Speicherplatzes, die Codegenerierung für indizierte Variablen[116], also „subscripted variables", die Ein-/Ausgabe. Besonders das automatische Ausführen der Indexierung durch den „translator" erzielte dabei den Autoren zufolge die größten Zeitersparnisse (Abschnitt Conciseness and Convenience, S. 43f.):

---

116 D. h. Variablen mit (in FORTRAN 1) bis zu drei Indizes, die also ein- bis dreidimensionale Matrizen darstellen konnten: z. B. A(I, K+1, J).

„The largest ratios [of coding times, assembly program language versus FORTRAN language] are usually obtained by those problems with complex loops and subscript manipulation as a result of the planning of indexing and bookkeeping procedures by the translator rather than by the programmer."

Der mit Abstand größte Teil der Instruktionen, die zur Programmierung des „translators" nötig waren, diente entsprechend der optimalen Behandlung von „subscripted variables", „DO-statements" und der „input-output" und „FORMAT statements" (ca. 18.500 Instruktionen) und nicht etwa der der arithmetischen Ausdrücke (ca. 5.500 Instruktionen). Denn (Abschnitt Method of Translation, S. 48): „The near-optimal treatment of arithmetic expressions is simply not as complex a task as a similiar treatment of ‚housekeeping operations'."

Der „translator" wird durch die gewählte Metaphorik in dem Text also als annähernd optimaler Haus- oder Buchhalter gezeichnet. Da er die Arbeit des Kodierens übernehmen soll, erscheinen weite Teile der Kodierarbeit im Gegenzug als Tätigkeiten, die gesellschaftlich eher niedrig bewertet sind, die aber grundlegend sind für weitere, als umfassender und anspruchsvoller geltende Arbeiten, wie z. B. die Buchhaltung für Entscheidungen des Managements in Unternehmen.

Im Gegensatz zum Kodieren erscheint zu Beginn des Textes die Entwicklung eines „generellen Plans" als kleine Aufgabe; das Programmieren in FORTRAN-Sprache scheint im wesentlichen in der Entwicklung dieses Plans zu bestehen (s. o., Abschnitt 4.4.2.3.1). Die Aufgabe der Vorbereitung wissenschaftlicher Probleme für einen Computer, die das FORTRAN-System erleichtern und reduzieren sollte (s. o., Abschnitt 4.4.2.2), erscheint in dem Text als bestehend aus zwei Teilen: einem untergeordneten und anspruchsloseren Teil, der anscheinend die meiste Zeit braucht und die Kodierung weitgehend ausmacht; demgegenüber steht ein übergeordneter und interessanterer Part, der durch die Programmierung repräsentiert wird.

Die Metaphorik – Buchhaltertätigkeit versus genereller Plan – kann als werbewirksame Strategie gesehen werden, das eigene Produkt in der Öffentlichkeit darzustellen. Zum einen richtet sich der Text weniger an Kodierer als vielmehr an deren Vorgesetzte, potentielle Käufer oder Wissenschaftler, die eher dem Management oder der mathematischen Problemformulierung verpflichtet waren, bevor sie an den Kodierer ging. Ihnen wird diese Metaphorik eingängig gewesen sein. Zum anderen schloß die FORTRAN-Gruppe mit dieser Wortwahl an einen Ansatz zum Computereinsatz in Wirtschaftsunternehmen an, der Mitte der 50er Jahre in den USA breit diskutiert und sehr schnell als Möglichkeit des Computereinsatzes akzeptiert wurde – den „Electronic Clerk Approach", wie ihn Theißing (1995) in Anschluß an Rose (1969) nennt. Nach diesem Modell sollte der Computer als „elektronischer Büroangestellter" dessen Arbeit – wie der „FORTRAN translator" die des Kodierers – besser, schneller und billiger ausführen, wobei „vor allem an die vielen schematischen Entscheidungen, die in den formalisierten Prozeduren der Verwaltung anfielen,

beispielsweise die Bestimmung von Überstunden anhand von Stechkarten" (Theißing 1995, S. 18), also einfache Haus- und Buchhaltertätigkeiten, gedacht wurde. Mit ihrer Metaphorik schlossen die Autoren also an eine Form des Computereinsatzes in Unternehmen an, die zu dieser Zeit auf breite Resonanz stieß.

Der „translator" wird – die Ausführungen dieses Abschnitts zusammengefaßt – in dem Text in ein Sinngeflecht eingebettet, das ihn als Übersetzer zwischen zwei Sprachen, als guten Kodierer und Buchhalter erscheinen läßt. Seine Operationen werden durch die Folie menschlicher Angestelltenarbeit interpretiert und wahrgenommen. Wie ein menschlicher Angestellter wird er einerseits als ausführend – als „executive routine" – gesehen; im Rahmen seines Aufgabengebiets, dem Übersetzen, scheint er jedoch weitgehend selbständig zu arbeiten und zu entscheiden. Der Begriff „translator", der in dem Papier fast ausschließlich benutzt wird (bis auf an drei Stellen, wo von dem „translator program" die Rede ist), läßt mehrere Interpretationen zu: Einerseits wird er als Ersatz des menschlichen Kodierers (und als Übersetzer zwischen Sprachen) verstanden; seine Operationen erscheinen vor diesem Hintergrund als Äquivalente menschlicher Handlungen. Andererseits wird er als Maschine, die mit dem „translator program" ausgerüstet ist, gesehen; und schließlich erscheint der „translator" auch selbst als Programm, als „executive routine for the 704". Dieses Oszillieren der Perspektiven entspricht der Hybridobjekt-Sichtweise: der „translator" kann als maschinell, als Symbolsystem oder als menschenähnlich gedacht werden.

4.4.3 Zusammenfassung: FORTRAN als automatisches Kodiersystem

Schon ein Jahr nach der ersten Auslieferung des FORTRAN-Systems im Jahre 1957 waren 60 von 66 IBM 704-Rechnern der Benutzerorganisation SHARE mit FORTRAN I ausgerüstet. Mehr als die Hälfte aller Maschinenanweisungen auf diesen Rechnern waren mit Hilfe von FORTRAN erzeugt worden. Während der Erfolg eines Kodiersystems in den 50er Jahren in der Regel von der Anzahl der installierten Rechner, auf denen es lief, abhängig war, war FORTRAN zu einem Verkaufsargument für die IBM 704 geworden, was sich an der für damalige Verhältnisse hohen Zahl an Neuinstallationen – 30 Stück – in den Jahren 1957 und 1958 zeigt. IBM antwortete auf diesen Erfolg mit der Anpassung von FORTRAN an andere IBM-Computer. 1958 stand FORTRAN für die IBM 705 und 709 zur Verfügung, welche, wie die IBM 704, Maschinen für technisch-wissenschaftliche Anwendungen waren.

Für die IBM 650, die bereits 1956 herausgekommen war, wurde schon wenige Monate nach Auslieferung des IBM 704 FORTRAN-Systems ein FORTRAN-Dialekt, FORTRANSIT, fertiggestellt. Die IBM 650 erfreute sich bereits relativ breiter Akzeptanz in der US-amerikanischen Wirtschaft: Sie war der erste massenhaft gefertigte

Computer; von ihr wurden 1800 Stück verkauft, womit IBM zum Marktführer im Computerbereich aufstieg. 1958 waren bereits für 265 IBM 650-Installationen FORTRANSIT Übersetzerprogramme angefordert (vgl. Backus 1958, S. 246f.). Beim Kauf der IBM 650 bewilligte die IBM 60 % „educational discount" für solche Institutionen, die die Maschine in der Ausbildung einsetzten. FORTRAN wurde so für viele Schüler und Studentinnen die erste Programmiersprache, die sie erlernten (vgl. Lee 1982, S. 22).

In den Folgejahren[117] wurde FORTRAN auch auf Rechnern anderer Firmen implementiert: als erstes 1961 für die UNIVAC Solid State 80 von Remington Rand. 1963 hatten so gut wie alle Rechnerhersteller eine Version von FORTRAN im Angebot oder waren zumindest dabei, eine vorzubereiten. Der Erfolg von FORTRAN war nicht mehr aufzuhalten. Bereits im Mai 1962 wurde ein Komitee der „American Standards Association" gegründet mit der Aufgabe, ein US-amerikanisches Standard-FORTRAN zu entwickeln. FORTRAN war damit die erste Programmiersprache, die durch eine entsprechende Organisation standardisiert wurde. Bis heute wird FORTRAN besonders im Bereich der Technik- und Naturwissenschaften sehr breit eingesetzt und gehört mit COBOL zu den am weitesten verbreiteten Programmiersprachen.

Mit der FORTRAN-Entwicklung wurde es üblich, vom „Programmieren" in einer „Sprache" zu reden. Programmieren in FORTRAN-Sprache schien zu bedeuten, einen generellen Plan für eine wissenschaftliche Problemlösung auf dem Computer in einer präzisen Sprache wie der der Mathematik zu erstellen. Da Programmieren in FORTRAN im wesentlichen in dem Erstellen einer mathematischen Problemlösung zu bestehen schien, konnte es leicht dem Aufgabenbereich von Ingenieuren und Wissenschaftlern zugeschlagen werden, denen auch zuvor die mathematische Problembeschreibung oblag. Für die „Verschlüsselung" dieser Beschreibung in Maschinensprache war ihnen zumeist ein professioneller Kodierer zugeordnet. Der „translator" sollte diesen Kodierer im wesentlichen ersetzen – und diese Botschaft kam an. 1962 erschien z. B. ein Buch mit dem Titel (Johnson, Smith 1962): „FORTRAN-Autotester. A self-training course designed to emancipate the scientist and engineer from the need for the professional programmer." Für den Anfang der 60er Jahre hielt Sammet (1969, S. 148) entsprechend fest:

> „During this period of time FORTRAN was becoming more and more widely used. In some sense, its introduction caused a partial revolution in the way in which computer installations were run because it became not only possible but quite practical to have engineers, scientists, and other people actually programming their own problems without the intermediary of a professional programmer."

---

117 Die weitere Entwicklung und Verbreitung von FORTRAN bis 1969 wird ausführlich behandelt in (Sammet 1969, S. 144ff.).

Die Reaktionen von Seiten der Kodierer auf FORTRAN fielen – nicht zuletzt auch entsprechend dem schlechten Bild, das die FORTRAN-Gruppe von ihnen zeichnete – wenig begeistert aus. Aus der Sicht von Backus, der 1980 seine Erinnerung in bezug auf Kodierer aufschrieb und diese dabei als exzentrische, dem Alkohol zusprechende „Priesterschaft" zeichnete, hört sich das so an (S. 128):

> „The priesthood wanted and got simple mechanical aids for the clerical drudgery which burdened them, but they regarded with hostility and derision more ambitious plans to make programming accessible to a larger population."

Aus der Sicht eines Kodierers, der 1960 ein Gedicht mit dem Titel „The Programmer Strikes Back" in der Datamation veröffentlichte, wird der Hintergrund der ablehnenden Haltung gegenüber FORTRAN deutlich: falls vom Kodierer überhaupt noch etwas übrigbleibt, dann ist er dem Gedicht zufolge jedenfalls nicht derjenige, der die Kontrolle über die Maschine ausübt. FORTRAN erscheint statt dessen als die kontrollierende Instanz (Ingrassia 1960):

> „Where once stood a proud and unconquerable soul
> There now stands a coder without core control
> A victim of FORTAN and all that it means
> This greedy new monster controls the machines."

FORTRAN wurde, wie das Gedicht deutlich macht, zumindest von einem Teil der Kodierer als entmündigend empfunden. Diese Empfindung entspricht der Darstellungsweise des besprochenen FORTRAN-Papiers: Der Kodierer erscheint als langsam und fehlerhaft, ein Großteil seiner Arbeit scheint Buchhaltertätigkeiten gleichzukommen. Subtiler drückt sich diese Entmündigung auch in der Begrifflichkeit aus, in der hier und auch in der Folgezeit über FORTRAN-Programme nachgedacht wurde: Während Kodierer in Maschinencode befahlen oder instruierten, machten Programmierer in FORTRAN-Sprache Aussagen oder gaben Erklärungen ab.

Dieser Änderung der Terminologie – von Instruktionen zu Aussagen – entspricht, daß der „translator" FORTRAN-Programme im Falle eines syntaktischen Fehlers zurückwies und in diesem Sinne als den Programmierer korrigierende Instanz fungierte. Auch im Falle eines korrekten Programms führte die FORTRAN-Maschine nicht einfach aus, sondern übersetzte das Programm in eine andere Sprache, die in dem Papier als dem Menschen als fremder und unverständlicher dargestellt wird als die FORTRAN-Sprache. Dabei wurde der aus einem FORTRAN-Programm generierte Maschinencode optimiert – und zwar von einer Instanz, die als selbständig handelnd, entscheidend, optimierend und produzierend wahrgenommen wurde, und ihre eigenen Entwickler in Erstaunen versetzte. Das schließlich laufende Programm schien im Gegensatz zu einem von einem Kodierer erstellten Programm nicht mehr nur das Produkt des Programmierers, sondern auch das des „translators" zu sein – so wie zuvor das Programm auch das Produkt des Kodierers war, den der „translator" ja ersetzen sollte. Der Programmierer schien so nicht mehr Herr seines Produkts zu sein; die Maschine programmierte gewissermaßen mit.

Zwar hatte ein selbst programmierender Wissenschaftler oder Ingenieur sich von dem menschliche Kodierer emanzipiert, mußte sich nun aber den Regeln der FORTRAN-Sprache und des „translators“ anpassen. Dabei schien es so, als käme das Programmieren in FORTRAN im wesentlichen dem Aufschreiben mathematischer Formeln gleich – als müsse nur das numerische Verfahren in einer quasi mathematischen Sprache spezifiziert werden. Was von Neumann und Goldstine noch als Differenz zwischen mathematischem Text und Code thematisiert hatten – daß ein Befehl nicht nur für sich selbst, sondern auch für alle Maschinenläufe durch den Befehl steht –, geriet dabei aus dem Blick. Das Programmieren als mathematische Tätigkeit zu sehen, war auch schon bei von Neumann angelegt. Daß nun das Programmieren darin zu bestehen schien, ein numerisches Verfahren in fast derselben Weise niederzuschreiben, wie das zuvor in der die Kodierung vorbereitenden Phase der mathematischen Problembeschreibung der Fall war, ließ das Programm als im wesentlichen mathematischen Text erscheinen. In den Hintergrund trat, daß beim Programmieren die Maschinenfunktion mitgeplant wurde. Mit dem Eindringen von FORTRAN in die Ingenieur- und Naturwissenschaften wurde, so die sich anschließende These, der zuvor mathematische Blick der nun selbst programmierenden Wissenschaftler und Ingenieure auf den Problembereich unbemerkt so modifiziert, daß die Erfordernisse der Rechenmaschine immer gleich mitgedacht wurden.

Wenn auch verhalten stellte die FORTRAN-Gruppe ihre Arbeit an dem FORTRAN-System 1957 auch als Beitrag zu Fragen der Sprachübersetzung allgemein dar (s. o., Abschnitt 4.4.2.3.2). Ein Jahr nach dem Erscheinen des oben besprochenen Papiers hielt Backus einen Vortrag über „automatic programming“ auf dem „Symposium on the Mechanization of Thought Processes“. Hier hielt er ähnlich fest (Backus 1958, S. 254):

> „[...] efforts to find better languages for stating procedures and better techniques for translating them into machine language will contribute to the general understanding of complex information-processing methods. Already various translators such as FORTRAN perform some of the more complex information-processing tasks presently done by machines.“

Das Entwickeln von Programmiersprachen und von verbesserten Techniken zu ihrer Übersetzung sollte Backus zufolge zum generellen Verstehen komplexer Informationsverarbeitungsmethoden beitragen. Der „FORTRAN translator“ wird in dem Zitat als informationsverarbeitend gesehen – er führt eine der komplexeren Informationsverarbeitungsaufgaben aus, die derzeit von Maschinen erledigt werden. Damit kann er zum Verstehen von Informationsverarbeitungsprozessen allgemein – nicht nur auf der Maschine – beitragen. Als Ersatz des Kodierers konzipiert konnte der „translator“ scheinbar auch über menschliche Informationsverarbeitung – zu der Sprachüber-

setzung ja gehört – etwas aussagen. Die Ähnlichkeit zwischen maschineller und menschlicher Informationsverarbeitung wird hier ohne Begründung, wie sie von Neumann für die Ähnlichkeit Nervensystem-Rechenmaschine zumindest partiell noch lieferte, angenommen. Deutlich wird, daß Sichtweisen und Ideen, die in der Regel als typisch für die später entstehende Disziplin der „Künstliche Intelligenz" (KI) angesehen werden, auch schon in der FORTRAN-Entwicklung eine Rolle spielten. Mit dem Anspruch, einerseits den menschlichen Kodierer durch den „translator" zu ersetzen und andererseits die Übersetzung zwischen „Sprachen" durchzuführen, war das FORTRAN-Projekt ein wichtiger Meilenstein auf dem Weg zur KI. FORTRAN wurde entsprechend recht früh auch für nicht numerische, typische KI-Anwendungen eingesetzt (Backus 1958, S. 247):

> „It is interesting to note the conveniences offered, and hence, the use of the system in the case of processes which are essentially non-numerical. For example, FORTRAN programs have been or are being written [...] to do the following: [...] play chess (J. McCarthy, MIT), [...] and prove theorems in geometry (H. Gelernter and N. Rochester, IBM)."

1962 benutzte ein Mitglied der ursprünglichen FORTRAN-Gruppe, David Sayre, bereits den Intelligenzbegriff, um das, was der „translator" tat bzw. auf jeden Fall tun sollte, zu beschreiben (Sayre nach Greenberger 1962, S. 274):

> „These circumstances naturally produce an intense pressure to economize on human intelligence in programming by developping machine intelligence as a substitute, so as to trade what is plentiful for what is scarce."

Was der „translator" tun sollte, den Kodierer weitgehend ersetzen, wurde von Sayre als Entwicklung maschineller Intelligenz gesehen. 1962 erschien der „translator" so als Beitrag zur Forschungsdisziplin „Künstliche Intelligenz", die sich in dieser Zeit herausbildete. In Abschnitt 4. 6 werde ich auf Programmiersprachenentwicklungen zu sprechen kommen, die dem Bereich der „Künstlichen Intelligenz" zuzurechnen sind und zum Teil, wie LISP, auf FORTRAN aufbauten. Zuvor wende ich mich den ersten Programmiersprachen zu, die anders als FORTRAN mit dem Ziel entwickelt wurden, maschinenunabhängig zu sein.

4.5 Maschinenunabhängige Programmiersprachen

Während die anderen in dieser Arbeit angesprochenen Bemühungen um die Programmierung von Rechenmaschinen von kleineren Personengruppen betrieben wurden, wurden ALGOL und COBOL von relativ großen Komitees entwickelt. Beide Sprachen sollten von speziellen Rechenmaschinen unabhängig sein und zudem in ihrem jeweils anvisierten Nutzungsgebiet eine Art Standard bereitstellen – ALGOL im Bereich der numerischen Berechnungen und COBOL im Bereich der wirtschaftlichen Datenverarbeitung. Bei beiden Sprachen wurde deshalb versucht, ein breites Spektrum Betroffener – Rechenmaschinenhersteller und Nutzer – in die Entwicklung der Sprache

miteinzubeziehen. Sehr anschaulich wurde die Situation, auf die beide Sprachen – auch ALGOL im wissenschaftlichen Bereich – antworteten, im Editorial der Zeitschrift „Management and Business Automation" vom März 1960 beschrieben:

> „The creation of a modern Tower of Babel is one unfortunate result of the spirited competition between computer manufacturers. Their efforts to maintain individuality of product have been responsible for the several dozen business language systems that confront and confuse today's computer users."

Da ALGOL und COBOL maschinenunabhängig sein sollten, stellte sich die Frage danach, wie diese Programmiersprachen eindeutig spezifiziert werden konnten, zum ersten Mal in dringlicher Weise. Denn die Sprachen waren nun nicht mehr durch Bezug auf den Rechner, für den sie entwickelt waren, definiert. Es mußte ein Weg gefunden werden, die zulässigen Sprachkonstrukte und ihre Bedeutung in eindeutiger Weise zu beschreiben. Nur so war möglich, daß verschiedene Rechenmaschinenhersteller Übersetzungsprogramme erstellen konnten, die das gleiche Programm auch in gleicher Weise verarbeiteten. Sowohl im Rahmen der ALGOL- wie auch der COBOL-Entwicklung wurden so Metasprachen entwickelt, die erlaubten, die Syntax einer Programmiersprache, die zulässigen Zeichen und Zeichenfolgen, exakt zu beschreiben.

Auch in bezug auf die formale Beschreibung der Semantik, die sich, wie die Geschichte zeigt, als erheblich schwieriger erwies, war man zunächst optimistisch. So hielt Backus, der im ALGOL-Komitee Vertreter der USA war, auf der „International Conference on Information Processing" in Paris fest (Backus 1959, S. 129):

> „The author had hoped to complete a formal description of the set of legal IAL programs and of their meanings in time to present it here. Only the description of legal programs has been completed however. Therefore the formal treatment of the semantics of legal programs will be included in a subsequent paper."[118]

Ich werde im folgenden kurz die ALGOL- und COBOL-Entwicklung besprechen. Auf den organisatorischen Ablauf, die verschiedenen Komiteetreffen und die beteiligten Personen gehe ich dabei nicht näher ein.[119] Im Zentrum meiner Überlegungen steht die Frage danach, wie ALGOL und COBOL mit dem Sprachbegriff in Verbindung gebracht und in welcher Weise sie als Sprachen verstanden wurden.

---

118 IAL steht hier für den ersten Namen von ALGOL, „International Algebraic Language", der jedoch recht bald von „ALGOL" abgelöst wurde.

119 Der Hergang der ALGOL-Entwicklung ist ausführlich beschrieben in (Perlis 1981), (Naur 1981), der der COBOL-Entwicklung in (Sammet 1981). Aus einer ähnlichen analytischen Perspektive wie ich nähert sich mein Kollege im Interdisziplinären Forschungsprojekt „Sozialgeschichte der Informatik", Peter Eulenhöfer (1998b, Kap. 3.3) der ALGOL-Entwicklung. In seiner Dissertation befaßt er sich mit der Entwicklung der Informatik in der Bundesrepublik Deutschland und damit auch mit ALGOL, das hier eine große Rolle spielte. Da sich unsere Ergebnisse sehr gut ergänzen, beziehe ich mich im Anschnitt zu ALGOL relativ ausführlich auf seine Arbeit.

4.5.1 Die „ALGOrithmic Language“

Die Entwicklung von ALGOL (ALGOrithmic Language) nahm ihren Anfang 1955 in Europa und 1957 in den USA. Während es den Europäern vor allem um eine elegante mathematische Sprache ging, war aus der Sicht der USA die Entwicklung einer maschinenunabhängigen Programmiersprache das zentrale Ziel[120] (Eulenhöfer 1998b, S. 130):

> „Während letztere [die europäischen Initiatoren] strukturelle Eleganz und Mathematiknähe der Sprache in den Vordergrund stellten, ging es den Computerbenutzenden [besonders der US-amerikanischen IBM-Benutzerorganisation SHARE] primär um ein Hilfsmittel zur Verwendung von Programmen auf verschiedenen Rechnertypen.“

Die ALGOL-Entwicklung hatte eine universale, internationale, algorithmische Sprache zum Ziel, die die europäischen und US-amerikanischen Anliegen in sich vereinen sollte: Sie sollte sowohl der Veröffentlichung von mathematischen Algorithmen als auch der Programmierung dienen. Der ALGOL-programmierte Rechner sollte also in der Lage sein, dieselben numerischen Algorithmen, die Wissenschaftler beim Lesen entsprechender Publikationen denkend nachvollzogen, zu verarbeiten. Die einzelnen ALGOL-Anweisungen wurden tendenziell als identisch mit den einzelnen mathematischen Denkschritten vorgestellt. Damit sollte endgültig der Prozeß der Übersetzung einer mathematischen Problemlösung in eine computergerechte Form wegfallen.

In der Schweiz war die Idee einer „algorithmischen Schreibweise“ als Grundlage der „automatischen Programmierung“ 1956 von Rutishauser ausgearbeitet worden. Die „algorithmische Schreibweise“ scheint bei ihm, wie später ALGOL, sowohl für die mathematische Problemformulierung als auch für die Programmierung geeignet zu sein; Mensch und Maschine können sie gleichermaßen verstehen. Zudem schreibt er ihr neben der beschreibenden auch operierende Kraft zu (Eulenhöfer 1998b, S. 122):

> „Erstens verwendet er den Terminus [algorithmische Schreibweise] in dem Sinne, daß ‚man das Problem in einer speziellen Form niederschreibt‘; zweitens gilt sie als ‚Methode zur Formulierung einer numerischen Berechnung‘; drittens heißt es: ‚Die algorithmische Schreibweise bewältigt auch komplizierte Rechenprozesse‘; viertens ist ihm wichtig, ‚daß die Maschine aus der algorithmischen Schreibweise das Programm selbst berechnen kann.‘ (Rutishauser 1956, S. 28f.) Damit gilt die algorithmische Schreibweise zugleich als Niederschrift eines Problems, als Formulierung einer numerischen Berechnung, als aktiver Bewältiger von Berechnungen, als etwas mechanisch Verarbeitbares und letztlich auf einer Maschine Ausführbares.“

ALGOL sollte – ähnlich wie bei Rutishauser die algorithmische Schreibweise – Programmier- und Kommunikationssprache für numerische Algorithmen sein. Sprachdesign war in der ALGOL-Entwicklung im Gegensatz zur FORTAN-Entwicklung der Hauptinhalt. Diskutiert wurden vor allem solche Formulierungsmittel, die den gedanklichen Ablauf betreffen, in den eine mathematische Problemlösung gefaßt wird, wie

---

120 FORTRAN kam als diese Sprache nicht in Frage, da es eine IBM-eigene Produktion war.

die Programmstruktur, das Prozedurkonzept oder der Geltungsbereich von Variablen. Nicht als primäre Probleme behandelt wurden hingegen Ein-/Ausgabeprozeduren, die die technischen Möglichkeiten der zu programmierenden Maschinen in den Vordergrund der Betrachtung rücken.

1959 formulierte Bauer – einer der Beteiligten aus Deutschland – die Aufgabe einer Konferenz zu ALGOL, die 1958 in Zürich stattgefunden hatte, wie folgt (Bauer 1959, S. 121):

> „Es war die Aufgabe der Konferenz, die arithmetische Schreibweise zu standardisieren und diejenigen Ergänzungen hinzuzufügen, die sie vollständig arbeitsfähig machen."

Bauer zufolge entspricht ALGOL also im wesentlichen der arithmetischen Schreibweise. Diese scheint verändert werden zu müssen, damit sie, scheinbar aus sich heraus, arbeitsfähig wird. Wie bei der FORTRAN-Entwicklung werden Schreibweisen aus der Mathematik und Programmiersprache als im wesentlichen identisch gedacht. Eulenhöfer (1998b, S. 137) hält in bezug auf den ALGOL-„Preliminary Report" von 1958 (Perlis, Samelson 1958) entsprechend fest: „Die ‚arithmetic statements' werden mit den gebräuchlichen arithmetischen Ausdrücken identifiziert."

Der Begriff der „Sprache" – weniger der der „Programmiersprache" – spielt in den ALGOL-Reporten eine zentrale Rolle. Allein in der zwei Seiten langen Einleitung des „Preliminary Report" zu ALGOL (Perlis, Samelson 1958) finden sich folgende Kombinationen über den Begriff „language" (nach Eulenhöfer 1998b, S. 135):

> „international algebraic language – algebraic programming language – computer language – programming language – formula language – problem language – problem oriented language – unified language – Reference Language – Publication Language."

Wie schon in der FORTRAN-Entwicklung wurden auch im Rahmen der ALGOL-Entwicklung Parallelen zwischen dem Übersetzen natürlicher Sprachen und dem von Programmier- in Maschinensprache gezogen. Auf der „International Conference on Information Processing" 1959 in Paris hielt Perlis fest (Perlis 1959, S. 153): „The algorithms [zum Übersetzen von Programmiersprachen in Maschinencode] are in fact close to those needed for translating natural languages".

Mit der durchgängigen Einführung der Sprachmetapher ging – wie bei FORTRAN – einher, daß der Maschine nicht mehr „Befehle" gegeben wurden. Statt dessen war von „statements" und „declarations" die Rede. Der Maschine sollte also nicht befohlen, sondern mitgeteilt und beschrieben werden. Ein Programm erschien als Sequenz von Ausdrücken bzw. Sätzen und Deklarationen (Eulenhöfer 1998b, S. 137):

> „Während zuvor Programme als Folgen von Befehlen gedeutet wurden, heißt es nun (Perlis/Samelson 1958): ‚Sequences of statements and declarations, when appropriately combined, are called programs.' "

Dabei erschienen die „statements" oder „expressions", wie sie – ähnlich wie später bei LISP – auch genannt wurden, zum einen als Regeln, die einen anonymen Akteur zum Handeln auffordern (Eulenhöfer 1998b, S. 138): „Die Beschreibung ist als Regel zu verstehen, als Aufforderung zum Handeln, zum Ausführen von Operationen. Wer diese

Regel befolgen soll, wird in dem Report nicht benannt." Gleichzeitig erscheinen die „statements", wie ich das bereits für die „FORTRAN-statements" aufgezeigt habe, auch selbst als die Kraft, die eine Aktion in Gang setzt (Eulenhöfer 1998b, S. 139):

> „An vielen Stellen legt die Sprechweise nahe, daß die Ausdrücke selbst diejenigen Instanzen sind, die eine Aktion verursachen. Sie werden zum Subjekt der Sätze und erscheinen als treibende Kraft des dynamischen Prozesses."

„Statements" wurden also, ähnlich wie in dem besprochenen FORTRAN-Text, als beschreibend und operierend konzipiert und konnten auch eine Handlungsaufforderung an einen anonymen Akteur, an Mensch, Maschine oder andere „statements", darstellen. Die „statements" werden als Äquivalent zu arithmetischen Ausdrücken und damit als Symbole gedacht; sie erscheinen als Beschreibungen, als Befehle und auch als selbst operierende Instanz.

Die Trennung der Hardwareebene von der Entwicklung der Sprache, die bei ALGOL zum ersten Mal durch die Forderung nach Maschinenunabhängigkeit und Verwendbarkeit als Publikationssprache nötig wurde, warf eine Fülle von neuen Problemen auf, die insbesondere die eindeutige Beschreibbarkeit der Sprache betrafen – Probleme, die zuvor deshalb nicht existierten, weil eine Programmiersprache bzw. ein Codiersystem nur in Zusammenhang mit einer bestimmten Maschine auftrat und durch den engen Bezug zur unterliegenden Technik klar beschrieben war. Die ALGOL-Entwicklung war so Ausgangspunkt für die internationale Beschäftigung mit Programmen und Programmiersprachen als Objekte, die man nun durch Bezug auf eine mathematisch-formale Ebene eindeutig zu fassen versuchte. Die ALGOL-Entwicklung und besonders der Versuch, eine maschinenunabhängige Programmiersprache zu erstellen, war damit ein Ausgangspunkt für die sich im folgenden etablierende „Theorie der Programmiersprachen". Charakteristisch ist für diese Disziplin die Konzeption von Programmen als mathematische Objekte; der Computer stellt nur einen Datenraum zur Verfügung, den die Programme verändern – als physische Maschine tritt er vollständig in den Hintergrund.[121]

4.5.2 Die „COmmon Business (O) Language"

1951 wurde eine UNIVAC an das statistische Bundesamt der USA zur Auswertung der Volkszählungsdaten von 1950 geliefert. Statt für wissenschaftliche Berechnungen, die viel Rechenzeit erforderten, aber wenig Eingabedaten benötigten und wenig Ausgaben produzierten, wurde damit eine elektronische Rechenmaschine dafür eingesetzt, Massendaten zu verarbeiten, d. h. einfache Berechnungen auf einer sehr großen Zahl

---

121 In (Eulenhöfer 1998b, Kap. 3.4) wird die Herausbildung der Theorie der Programmiersprachen und die damit verbundenen Versuche, die Semantik von Programmiersprachen formal zu fassen, genauer untersucht.

strukturierter Daten auszuführen. Ab Mitte der 50er Jahre wurden Rechner – von ihren Herstellern als billige „elektronische Angestellte" gepriesen – für die Verarbeitung von Massendaten wie Lohnbuchhaltung, Scheckverbuchung und Flugreservierung auch in Wirtschaft und Verwaltung eingesetzt. Es entstanden DV-Abteilungen in den verschiedenen Firmen, da diese ihre Anwendungen in der Regel selbst programmierten. Wie eine Analyse der US-amerikanischen Management-Zeitschrift „Harvard Business Review" sowie der Computerzeitschrift „Datamation" im Rahmen des Forschungsprojektes „Sozialgeschichte der Informatik" ergab (Theißing 1995), wurden diese Abteilungen von Anfang an als schwer zu kontrollieren und einseitig auf die technischen, statt auch die Probleme des Einsatzbereiches ihrer Programme ausgerichtet beschrieben. Häufig wurde die unterschiedliche Sprache von Programmierern und Fachleuten beklagt (vgl. Abschnitt 4.2).

Hier setzte die Entwicklung von FLOWMATIC für die UNIVAC-Rechenmaschinen unter Leitung von Grace Hopper an. Da für kommerzielle und wirtschaftliche im Gegensatz zu mathematischen Anwendungen keine einheitliche Terminologie existiere – so Hopper 1958 in einem Vortrag über „automatic programming" auf dem „Symposium on the Mechanization of Thought Processes" – und zudem Datenverarbeitungssysteme „alpha, alpha-numeric, and numeric quantities, with and without decimal points in various positions" (Hopper 1958, S. 171) verarbeiten müßten, sei zum einen nötig, neue Methoden zur Beschreibung von Daten zu entwickeln. Zum anderen müsse eine einheitliche Sprache oder Symbolik für die Vereinbarung der Operationen erarbeitet werden. Die einzig mögliche Grundlage für eine solche Symbolik bot nach ihr die englische Sprache (Hopper 1958, S. 171):

> „It was determined that the only common symbology used by the various installations was the English language and it was therefore decided to use a selected English vocabulary as a pseudo-code."

Die FLOWMATIC-Gruppe extrahierte deshalb aus der englischen Sprache dreißig Verben, die nach ihrer Meinung die Beschreibung von Datenverarbeitungsproblemen in einer gewohnten Weise möglich machten. Nach Austesten des FLOWMATIC-Compilers waren die Reaktionen insbesondere auf Seiten der „Air Material Command" euphorisch und man hoffte, durch die Einführung von „English language compilers" sowohl die Programmierzeit zu verkürzen als auch die schwerwiegenden Verständigungsprobleme zwischen Management und Programmierabteilung zu lösen. Colonel Miller hielt als einen der Hauptvorzüge von FLOWMATIC fest (nach Hopper 1958, S. 180):

> „Communication between all offices will be enhanced by use of this English Type Pseudo-Code, which largely removes the barrier now existing between operating management and computer programming groups."

Der Optimismus, der hier bezüglich der Verständlichkeit von „englischsprachigen" Programmen durchscheint, war bei der Entwicklung von COBOL (Common Business

Language) ungebrochen, die 1959 in den USA begann. Entwickelt werden sollte eine gemeinsame Wirtschaftssprache für „automatic digital computers". Treibende Kraft war das Pentagon, das – als größter Computernutzer in den USA – 1960 ankündigte, nur noch Rechenmaschinen zu kaufen oder zu mieten, für die COBOL-Compiler zur Verfügung standen.[122] Mit dieser Ankündigung war die Voraussetzung dafür geschaffen, daß COBOL sich als De-facto Standard in der wirtschaftlichen Datenverarbeitung durchsetzen konnte.

Oberste Maxime, an der sich die COBOL-Entwicklung orientierte, war „naturalness". Wie wörtlich im Rahmen der Diskussion um COBOL diese „Natürlichkeit" zum Teil genommen wurde, verdeutlicht ein kritischer Kommentar zu COBOL von 1960 (Bremer 1960, S. 22):

> „The extreme difficulties of developing such a language in a short period of time cannot be overemphasized. One has only to look at the gradual evolution of English to see what thousands of options have sprung up and then disappeared.
>
> Although it is desirable to base a business language upon a natural language such as English [...] there are nevertheless certain restrictions of present day computers which make the variety in English undesirable. For example, we understand a man who speaks English even though he stutters, but this is not economical to expect the machine to decode stuttering."

Da die Sprache im Gegensatz zu FORTRAN oder ALGOL nicht der mathematischen, sondern der natürlichen Sprache ähneln sollte, wurde vermieden, daß die Ausdrücke in irgendeiner Weise mathematischen Formeln glich. So wurden „arithmetische Verben" – ADD, SUBTRACT, MULTIPLY, DIVIDE – an Stelle der entsprechenden mathematischen Zeichen eingeführt (vgl. Sammet 1981, S. 205). In Anlehnung an den Aufbau der englischen Sprache begannen einzelne COBOL-Anweisungen wie imperative englische Sätze mit dem Verb. Es waren lange Variablennamen zugelassen und „noise words" – d. h. Worte, die syntaktisch nicht nötig waren, aber die Lesbarkeit erhöhten – konnten eingefügt werden.

Auch die Terminologie, in der über COBOL geredet wurde, lehnte sich eng an das Vorbild der natürlichen Sprache an. Exemplarisch dazu ein Ausschnitt aus einer Sprachbeschreibung von Jean Sammet, eine der wichtigsten Entwicklerinnen von COBOL (Sammet 1961, S. 205f.):

---

122 Einen Eindruck von dem Einfluß des „Department of Defense" und auch von den Kosten, die diesem durch die Computernutzung entstanden, gibt folgender Auszug aus dem Editorial der Zeitschrift „Management and Business Automation" vom März 1960: „According to his [gemeint ist Charles A. Phillips, Direktor des ‚Data System Research Staff' des ‚Department of Defense'] figures the department has some 225 general pupose computer systems in action with about 175 more on order. In addition, the civilian agencies of the government have installed about 60 systems with approximately 30 more in order. The cost of programming these business systems exceeded $15 million in 1959 and will be about $29 million in the current fiscal year. [...] It is apparent from these facts and figures why the Department of Defense has a vital interest in developing a common language".

„Verbs may be grouped together to form statements which in turn may be grouped together to form sentences. The latter are grouped to form paragraphs which in turn may be grouped to form sections."

Die COBOL-Maschine, die eine Art natürliche Sprache zu verstehen schien, erschien als eine, mit der man „kommunizierte". Häufiger als in den bisher betrachteten Sprachentwicklungen wurde von der „Kommunikation" zwischen Mensch und Maschine geredet, so z. B. von Phillips (1960), der das COBOL-Projekt als „joint effort of computer users and manufacturers, to improve communications between men and machines" charakterisierte. Neben der Kommunikation mit der Maschine sollte COBOL wie schon ALGOL auch der zwischen Menschen – Management und Programmierabteilung – dienen. Entsprechend den beklagten Kontroll- und Verständigungsproblemen bezüglich der DV-Abteilungen galt das Hauptaugenmerk der Entwicklerinnen und Entwickler weniger der leichten und schnellen Schreibbarkeit als vielmehr der Lesbarkeit der späteren COBOL-Programme. Daß in bezug auf die leichte Verstehbarkeit jedoch schon sehr bald Zweifel laut wurden, zeigt die Klage eines DV-Managers Anfang der 60er Jahre (Hughes nach Greenberger 1962, S. 285): „My programmers are using English language compilers to write their programs, but looking over their work I still don't know what they are doing."

ALGOL wie auch COBOL wurden als „universal language" eingeführt, wobei sich die Charakterisierung als universell in der Regel auf einen bestimmten Bereich – den der numerischen Berechnungen bzw. der wirtschaftlichen Datenverarbeitung – bezog. Doch der Anspruch einer universellen Sprache wurde auch in einem weiteren Sinne erhoben. Im Rahmen der COBOL-Entwicklung wurde ein Gruppe eingerichtet, die „Long-Range Task Group"; längerfristig sollte diese auf eine universelle Sprache hinwirken, die beide Bereiche abdeckte (Phillips 1960, S. 6):

„The Long-Range Task Group was to explore fundamentals and philosophies of computer languages, wether used for scientific or business-type problems, and look into the feasibility of developing a Universal Computer Language which might combine ALGOL with the new business oriented language."

4.6 Die „Information Processing Language" und der „LISt Processor"

Während mit FORTRAN und ALGOL vor allem Berechnungen durchgeführt werden sollten, sollten mit Hilfe von IPL und LISP Denk- und Problemlösungsprozesse allgemein programmiert werden. Um auch Daten nicht-numerischer Natur mit variabler Länge verarbeiten zu können, führten Newell, Shaw und Simon 1956 Listen ein, eine Datenstruktur, die z. B. Worte oder ganze Sätze aufnehmen kann. 1969 schätzte Sammet diese Neuerung wie folgt ein (S. 388f.): „One of the most significant events that has ever occurred in programming was the development of the concept of list processing by Allen Newell, J. C. Shaw and Herbert Simon." Die von diesen dreien

entworfene Programmiersprache IPL (**I**nformation **P**rocessing **L**anguage) war die erste listenverarbeitende Programmiersprache und wurde von einem relativ großen Personenkreis benutzt (Wegner 1976, S. 1212):

> „IPL V was an important milestone both because it was widely used for a period of over ten years by an important segment of the artificial intelligence community and because it pioneered many of the basic concepts of list processing."

IPL war die erste recht breit eingesetzte Programmiersprache, die auch in ihrem Namen explizit als „Sprache" bezeichnet wurde – ein Umstand, der Simon bis heute nicht aufgefallen war, wie in einem Interview, das ich im September 1994 mit ihm führte, deutlich wurde.

IPL gilt als der wichtigste Vorläufer des bis heute bekannten und angewandten LISP (**LIS**t **P**rocessor)[123], das erst Mitte der sechziger Jahre die Rolle von IPL als führende Sprache in der „Artificial Intelligence" übernahm. Zwar unterscheidet sich die äußerer Form der beiden Sprachen erheblich, jedoch finden sich zentrale Konzepte aus IPL – wie besonders die Darstellung von Daten und auch Programmen als Listen sowie die Art ihrer Verarbeitung – in LISP wieder. Vor Beginn der Implementierung von LISP war neben IPL noch eine andere listenverarbeitende Programmiersprache, FLPL (**F**ORTRAN **L**ist **P**rocessing **L**anguage), die ebenfalls an dem Listenkonzept von IPL orientiert war, bereits lauffähig. Von FORTRAN bzw. FLPL übernahm LISP besonders auch die Idee einer mathematisch orientierten Notationsform.

Ich werde im folgenden auf die Entwicklung von IPL und anschließend von LISP ausführlicher eingehen. In Erweiterung des FORTRAN-Vorhabens, den menschlichen Kodierer mit Hilfe eines Programms, des „translators", zu ersetzen, traten beide Sprachen mit dem Ziel an, menschliches Denken allgemein zu programmieren. Beide Sprachen spielten in der Forschungsdisziplin „Künstliche Intelligenz" eine zentrale Rolle und ihre Entwickler, besonders Newell, Simon und McCarthy sowie auch Minsky, zählen zu den exponiertesten Vertretern dieses Gebiets. Bei der Besprechung der zu IPL und LISP verfaßten Texte zeigt sich, daß ihnen ähnliche Orientierungsmuster unterliegen, wie den in Kapitel 2 und 3 besprochenen Texten: Mensch, Maschine und Symbolsystem werden eng zusammengedacht; auf Grundlage behavioristischer Vorstellungen scheint eins für das jeweils andere stehen zu können.

Bevor ich mit der Besprechung von IPL beginne, kontrastiere ich kurz die IPL und LISP-Entwicklung mit der FORTRAN-Entwicklung. Auf eine Gegenüberstellung mit der ALGOL- und COBOL-Entwicklung verzichte ich weitgehend, weil ALGOL und COBOL in einem völlig anderen Setting – in Komitees – entwickelt wurden. Ich zeige auf, daß IPL und LISP im akademischen Umfeld entstanden und fast bedingungslos und großzügig finanziert wurden. Weder die IPL- noch die LISP-Entwickler sprachen

---

123 Heute wird LISP in der Regel als Akronym für „LISt Processing Language" gehandelt; in den frühen Texten wurde es von McCarthy jedoch als Abkürzung für „LISt Processor" eingeführt.

in ihren Texten die breite Masse der mit Computern Beschäftigten an. Die Autoren präsentierten sich als Forschungsspitze; das Programmieren mit IPL und LISP erscheint entsprechend als intellektuelle Herausforderung.

Im Vergleich mit der FORTRAN-Entwicklung fallen sowohl für IPL als auch LISP verschiedene Differenzen sofort ins Auge:

Kosten- und damit zusammenhängend Geschwindigkeitsüberlegungen spielten in den IPL- und LISP-Texten keine Rolle: Nirgends wurde versucht, die Vorteile der eigenen Entwicklung zahlen- oder kostenmäßig zu fassen. Im Gegenteil: Es wurde sogar zugegeben, daß die Sprachen in der Anwendung sehr aufwendig und damit kostspielig war. So heißt es in einem IPL-Text (Newell, Shaw 1957, S. 232): „The present language has a number of shortcomings. It is very costly both in memory space and time". In dieser relativen Sorglosigkeit in bezug auf Speicherplatzverbrauch und Laufzeit der Programme spiegelt sich ein zentraler Unterschied zwischen der Finanzierung der FORTRAN- und der IPL- bzw. LISP-Entwicklung wider: Während FORTRAN – wenn auch großzügig finanziert – der Kosten-Nutzen-Logik eines wirtschaftlichen Unternehmens unterlag, wurden IPL und LISP im akademischen Bereich und mit erheblicher Unterstützung des Militärs finanziert – allerdings ohne daß damit spezifische Zielsetzungen verbunden waren. So wurde das „MIT[124] Artificial Intelligence Project", wo LISP entwickelt wurde, vom „MIT Research Laboratory of Electronics" unterstützt, das seinerseits über große Geldsummen verfügen konnte, die ihm formlos vom Militär zur Verfügung gestellt wurden. Sehr anschaulich beschrieb McCarthy die Finanzierungsgepflogenheiten bei einem Vortrag zur Geschichte von LISP (McCarthy 1981, S. 185):

> „[...] the way it took place is that Marvin Minsky and I asked Jerry Weisner, who was the head of the Research Laboratory of Electronics, if we could have an Artificial Intelligence project. [...] And he said, ‚Yes, what do you need?' And we said, ‚We need two programmers, a secretary, and a keypunch.' And he said, ‚Well, how about taking also six graduate students?' And we said, ‚Oh yes – graduate students, too.' "

Entsprechend der großzügigen und relativ bedingungslosen Finanzierung waren IPL und LISP nicht darauf angewiesen, ein großes Publikum anzusprechen. Die Sprachen wurden nicht zuletzt als Mittel der Forschung gesehen, die als solche nicht einfach und breit nutzbar sein mußten. So heißt es zu IPL (Newell, Shaw 1957, S. 232): „The language is purely a research tool, developed for use by a few experienced people who know it very well." Die zugehörigen Texte sind entsprechend im Vergleich zu dem FORTRAN-Text schwerer zu lesen und haben häufig lange formale Passagen, die nur mathematisch geschulten Lesern zugänglich sind. Auch der Dokumentation in Form von Handbüchern, Primern etc. wurde weit geringere Bedeutung zugemessen, als das in der FORTRAN-Entwicklung der Fall war. Zudem wurde erheblich weniger Wert

---

124 MIT steht für „Massachusetts Institute of Technology".

auf die Behandlung einer vereinfachten Ein- und Ausgabe gelegt. Entsprechend schätzte Sammet 1969 IPL als eine Sprache nur für professionelle Programmierer ein. LISP charakterisierte sie als sehr schwierigere Sprache (Sammet 1969, S. 388, S. 408, S. 416):

> „It is extremely difficult to read and write [...]. Depending on the mental set and receptivity of the audience it is either easy or hard to learn. [...] LISP is definitely *not* intended for use by either a novice programmer or a nonprogrammer; it requires a certain amount of sophisticated background to appreciate and use the language effectively. [...] its use has been limited to a relatively few diligent people who can cope with the notational inconveniences."

Noch 1993 beschrieben Steele und Gabriel – beide versierte LISP-Programmierer – unter der Fragestellung „Why LISP is diverse" in der Rubrik „Its people" diese als „adventuresome", „exploring the most difficult and exotic of all programming tasks: mimicking or understanding cognition and intelligence", „generally creative and bold" und LISP selbst als „language of experts" (Gabriel, Steele 1993, S. 67ff.). Während FORTRAN, das in Folge von ALGOL entwickelte PASCAL und COBOL bei US-amerikanischen Studenten und Hackern später eher als uninteressant eingeschätzt wurden, erscheint LISP als aufregend und ungewöhnlich (Turkle 1984, S. 280):

> „FORTRAN und COBOL oder die ‚wissenschaftliche' Sprache PASCAL sind zum Symbol der Uniformität einer Massenkultur geworden, die das Individuelle in der Menge untergehen läßt. [...] LISP dagegen ist die Sprache der Lust, der Individualität."

Diese Einordnung der Sprachen spiegelt wohl nicht zuletzt wider, daß – während FORTRAN als einfach und leicht zu lernen dargestellt wurde – IPL und LISP sich an den forschenden und mathematisch versierten Wissenschaftler richteten, der unter Umständen selbst programmierte oder aber nah an der Programmierung arbeitete. Der Programmierer erschien als jemand, der in der Lage ist, mit komplizierten Problemen fertig zu werden und eine schwierige Sprache zu beherrschen.

Die Autoren selbst redeten von sich als „we" (wie in der Wissenschaft allgemein üblich auch McCarthy, der zumeist alleine schrieb). Sie erscheinen in ihren Texten als Personen, die sich mit Fragen auseinandersetzen, die die Spitze der aktuellen Forschung ausmachen. So wurde z. B. in einem IPL-Text der „Logic Theorist" (LT) – das Programm, das mit IPL programmiert werden sollte (s. u.) – wie folgt eingeführt (Newell, Shaw 1957, S. 230): „LT is representative of a large class of problems which are just beginning to be amenable to machine solution;" 1959 führte McCarthy ähnlich LISP als System ein, mit dem Allgemeinwissen programmiert werden kann (McCarthy 1960, S. 203) :

> „The system was designed to handle experiments with a proposed system called the Advice Taker, whereby a machine could be instructed to handle declarative as well as imperative sentences and could exhibit ‚common sense' in carrying out its instructions."

Wie das für Abhandlungen naheliegt, die mit neuen Gebieten befaßt sind, wurden in den Texten häufig Visionen entworfen und als offen und kompliziert dargestellte Fragestellungen angedacht, ohne vollständig gelöst zu werden. Die Autoren präsen-

tierten sich damit als Visionäre, die neue Forschungsgebiete erschließen und hier Pionierarbeit leisten.

Von der Maschine ist in den IPL- und LISP-Texten eher selten die Rede; wenn doch, dann ist zumeist die mit einem Programm – dem „Logic Theorist" bzw. dem „Advice Taker" – ausgerüstete Maschine gemeint. Der JOHNNIAC-Computer für IPL bzw. die IBM 704 für LISP erscheinen, zumindest falls entsprechend programmiert, als Instanz, die in der Lage ist, zu lernen, zu verstehen, Probleme zu lösen, zu entscheiden usw. Die potentielle Fähigkeit der Maschine, im wesentlichen so zu denken wie ein Mensch, wird dabei mehr oder weniger selbstverständlich vorausgesetzt. Die Maschine erscheint als ein Gerät, das im intellektuellen Bereich im wesentlichen das gleiche wie der Mensch zu leisten im Stande ist.

4.6.1 IPL

IPL wurde ab 1956 von Newell, Shaw und Simon entwickelt. Der Programmiersprache IPL, die aus heutiger Sicht auf Grund der Einführung des Listenkonzeptes recht große Bedeutung gewinnt, widmeten die Autoren zunächst keine eigene Veröffentlichung, die im Titel den Namen der Sprache trug. Vielmehr hießen ihre im folgenden auch besprochenen Texte:

- „The Logic Theory Machine - A Complex Information Processing System" (1956), in dem sie ihre Sprache noch „Logic Language" (LL) nannten;
- „Empirical Explorations with the Logic Theory Machine: A Case Study in Heuristics" (1957);
- „Programming the Logic Theory Machine" (1957).

Erst 1960 erschien der Artikel „An Introduction to Information Processing Language V" von Newell und Tonge. Die Programmierung des „Logic Theorist" (LT) stand also zunächst im Zentrum der Bemühungen um IPL. Ich werde deshalb nach einer kurzen Beschreibung der Vorgeschichte auf den LT näher eingehen und aufzeigen, daß der LT – verstanden als Beispiel für Informationsverarbeitungs- und Problemlösungsprozesse allgemein – dazu dienen sollte, Informationsverarbeitungsprozesse auf dem Computer statt im Labor empirisch zu erforschen. Wie ein Mensch schien er auf Grundlage von Faustregeln zu entscheiden und in nicht vorhersehbarer Weise Probleme eigenständig zu lösen (Abschnitt 4.6.1.2). IPL, die Sprache mit welcher der LT realisiert werden sollte, sollte erlauben, solche flexiblen, im Vorhinein scheinbar nicht vollständig beschreibbaren Prozesse zu programmieren (Abschnitt 4.6.1.3). Das Listenkonzept, das ermöglichte, Speicherplatz zur Laufzeit eines Programms flexibel zuzuteilen, spielte für die Einschätzung, mit IPL eine solche Aufgabe erfüllen zu können, eine zentrale Rolle. Neben der Programmierung sollte IPL erlauben, die Charakteristika eines Informationsverarbeitungssystems theoretisch-formal zu fassen. Programme

erschienen so als psychologische Theorien. Abschließend (Abschnitt 4.6.1.4) fasse ich die Ergebnisse zusammen und argumentiere, daß eine behavioristisch-atomistische Sichtweise, wie sie in von Neumanns Texten deutlich wurde, auch der IPL-Entwicklung unterlag.

4.6.1.1 Vorgeschichte

Allen Newell (geb. 1927) und Herbert Simon (geb. 1916) lernten sich 1952 bei der „RAND Corporation" kennen. Simon wurde hier als Berater in einem Projekt eingestellt, in dem Newell und andere daran arbeiteten, das Verhalten einer Luftverteidigungs-Frühwarnstation mit ca. 50 Mitarbeitern zu untersuchen, die im „Systems Research Laboratory" simuliert wurde (Simon 1991, S. 168):

> „[...] four people – John Kennedy, William Biel, Robert Chapman, and Allen Newell [...] – conceived the grand, or grandiose, design of studying in the laboratory the behavior of an air defense organization [an air-defense early warning station, wie es im nächsten Satz erläuternd heißt]."

Im Rahmen der Forschung der U.S. Air Force wurde über Luftverteidigungsstationen schon 1950 als „Organismen" nachgedacht, die wie andere Organismen auch handeln und entscheiden. Während bei von Neumann (1948, s. o. Abschnitt 3.1) Automaten Organismen und Maschinen sein konnten, erschienen hier Organismen als biologisch, als maschinell oder auch als biologisch-maschinelle Mischformen (aus dem „Progress Report of the Air Defense Systems Engineering Committee" vom Mai 1950; nach Keller 1995, S. 90):

> „The Air Defense System then is an organism [...]. What then are organisms? They are of three kinds: animate organisms which comprise animals and groups of animals, including men; partly animate organisms which involve animals together with inanimate devices such as the ADS; and inanimate organisms such as vending machines. All these organisms posses in common: sensory components, communication facilities, data analyzing devices, centers of judgement, directors of action, and effectors, or executing agencies. [...] It is the function of an organism to interact with and alter the activities of other organisms, generally to achieve some defined purpose."[125]

Konzipiert als Organismen konnte also nicht nur das Verhalten bzw. Handeln von Maschinen und Menschen, sondern auch von Organisationen, die mehrere Menschen

---

125 Die Rechenmaschine so wie von Neumann sie 1945 beschrieb (vgl. Abschnitt 2.2) kann auch nach dieser Definition als Organismus gesehen werden: Die „sensory components" entsprechen den Eingabeorganen (nach von Neumann die Äquivalente der „sensory neurons"); die „communication facilities" den Leitungen, auf denen Daten zwischen den verschiedenen Organen des Geräts transportiert werden; die „data analyzing devices", „centers of judgement" und „directors of action" sind in dem „Central Control Organ" vereint, das Befehle aus dem Speicher analysiert, beurteilt und die entsprechenden Operationen der anderen Organe veranlaßt; die „effectors, or executing agencies" entsprechen der „Central Arithmetic", die die Anweisungen der „Central Control" ausführt, und den Ausgabeorganen (bei von Neumann die Äquivalente zu den „efferent neurons").

und Maschinen umfaßten, in gleicher Weise untersucht werden. In von Neumanns Automatentheoriepapier (von Neumann 1948) stellte das Zusammenwirken des Verhaltens der elementaren Elemente die Ebene dar, auf der Maschinen und lebendige Organismen vergleichbar schienen. In dem Zitat ist es das Zusammenwirken der dort aufgeführten Komponenten, welches die gemeinsame Untersuchungsebene für die verschiedenen biologischen und/oder maschinellen Organismenarten darstellt. Ein Organismus wird gedacht als bestehend aus sensorischen Komponenten, Einrichtungen für die Kommunikation, die Analyse von Daten und deren Beurteilung, Direktoren der Aktion und ausführenden Einheiten. Ein Organismus erscheint als Einheit, die Daten aus der Umgebung erspürt, verarbeitet und Aktionen ausführt, die einem vorgegebenen Ziel dienen. Einen Organismus als informationsverarbeitendes System zu sehen – wie bald darauf Newell und Simon – ist von hier aus nur ein kleiner Schritt.

Als Simon in dem „Systems Research Laboratory" der „RAND Corporation" eingestellt wurde, hatte er bereits eine Professur für „Administration" (dtsch. Verwaltung) an der Carnegie-Mellon-Universität in Pittsburgh inne und war dort Leiter des „Department of Industrial Management". 1945 hatte er ein sehr erfolgreiches Buch, „Administrative Behavior" (deutscher Buchtitel: „Das Verwaltungshandeln"), eine Überarbeitung seiner PhD-thesis, veröffentlicht. Hier beschäftigte Simon sich mit der Frage, wie eine Verwaltung oder Organisation Entscheidungen trifft. Die Terminologie, derer er sich dabei bedient, z. B. „Verwaltungsbiologie" oder „Anatomie und Physiologie der Organisation" (Simon 1945, S. 142), macht deutlich, daß er Verwaltungen und Wirtschaftsunternehmen als eine Art Organismen auffaßte – ähnlich wie oben Luftverteidigungsstationen als Organismen erscheinen. Wie der Organismus Mensch zu handeln und Entscheidungen zu treffen in der Lage ist, betrachtete er Verwaltungen und Wirtschaftsunternehmen als entscheidende und handelnde Instanzen. Sein Entscheidungs- und Handlungsbegriff war also nicht an den einzelnen Menschen gebunden.

In dem Buch erscheinen Entscheidungen als Wahl unter einer Reihe von Möglichkeiten; entsprechend werden die Ausdrücke „choice" und „decision" meist synonym verwendet (vgl. Simon 1945, S. 3). Eine rationale Entscheidung zeichnet sich dadurch aus, daß eine Möglichkeit ausgewählt wird, die zur Erreichung eines zuvor gewählten oder bestimmten Zwecks oder Ziels dient. Dabei war für Simon (Simon 1945, S. 4)

> „jede Entscheidung ein Kompromiß. Die schließlich gewählte Möglichkeit läßt niemals eine vollständige oder vollkommene Verwirklichung der Ziele zu, sondern ist nur die unter den gegebenen Umständen erreichbare beste Lösung."

Für Simon war Entscheiden nicht unbedingt ein logisch-deduktiver Prozeß mit dem einen, optimalen Ergebnis. Als Wahl unter gegeben Möglichkeiten sollte eine Entscheidung lediglich zu einem möglichst guten Ergebnis führen. Seine 1945 entwickelte Sicht auf das Entscheiden lag später auch dem Konzept der heuristischen Programmierung, das Newell, Shaw und Simon in die Programmierung einführten,

zugrunde (s. u.). Daß er Organisationen, die Menschen und Maschinen umfaßten, als ein Art Organismen dachte, paßt – wie auch der oben dargestellte Organismenbegriff verdeutlicht – dazu, daß für Simon die Rechenmaschine später ganz selbstverständlich als denkende und intelligente Instanz galt (s. u.).

Simons Erfahrungen im Bereich der Organisationsberatung waren 1952 der Grund, aus dem er vom „Systems Research Laboratory" der „RAND Corporation" als Berater angestellt wurde (vgl. Simon 1991, S. 168). Wie die Air Force hier eine Luftverteidigungsstation hatte Simon zuvor Verwaltungen und Unternehmen als Organismen untersucht. Im „Systems Research Laboratory" konnte Simon seinem Interesse an Entscheidungsprozessen weiter nachgehen. Mit Newell zusammen versuchte er, diese Prozesse formal zu beschreiben. Die dreijährigen Experimente lieferten zwar viele sehr genaue Daten darüber, wie Radar-Operateure und Luftraumkontrolleure ihre Entscheidungen fällten, eine formale Beschreibung gelang Simon und Newell jedoch nicht (Simon 1991, S. 168):

> „Yet Al and I suffered from continuing frustration in trying to write formal descriptions of the process. Somehow, we lacked the necessary language and technology to describe people as information processors."

Im Rahmen der Experimente des „Systems Research Laboratory" programmierten Cliff Shaw, nach Simon „an outstandingly talented and sophisticated systems programmer", der eher bescheiden und schweigsam war (vgl. Simon 1991, S. 201, 211), und Newell auch Imitationen von Radarkarten für die Simulation der Luftverteidigung. Dabei produzierte der Computer nicht nur Zahlen, sondern Punkte auf einer zweidimensionalen Karte. Langsam wurde Newell und Simon daraufhin klar – so Simon in seiner Biographie –, daß der Computer nicht nur Zahlen, sondern Symbole allgemein verarbeiten und damit den Formalismus zur Beschreibung von Verhalten bieten könne, den Simon und Newell ja suchten (Simon 1991, S. 201):

> „This insight, which dawned only gradually, led Al and me even more gradually to the idea that the computer could provide the formalism we were seeking – that we could use the computer to simulate all sorts of information processes and use computer languages as formal descriptions of those processes."

4.6.1.2 Der „Logic Theorist"

In der Folgezeit veröffentlichten Newell, Simon und Shaw verschiedene Papiere, die sich dem Problem der formalen Beschreibung des Entscheidens und Problemlösens mit Hilfe des Computers befaßten. Auf der „Western Joint Computer Conference" von 1955 stellte Newell das Papier „The Chess Machine[126]: An Example of Dealing with a

---

126 Dabei war die Schachmaschine der Rechner ausgerüstet mit dem Programm (Newell 1955, S. 101): „From now on the computer with program will be called the ‚machine', and the various parts of what it does will be called ‚mechanisms'."

Complex Task by Adaption“ vor. Das Schachproblem war für ihn ein Fallbeispiel, das für eine Klasse „ultrakomplexer“ Probleme stand, wie das Übersetzen von Sprachen oder das Zusammenfassen wissenschaftlicher Artikel (vgl. Newell 1955, S. 101).

Im September 1956 hielten Newell und Simon auf dem „Symposium on Information Theory“ den Vortrag „The Logic Theory Machine. A Complex Information Processing System“. Hier stellten sie das erste Mal die „logic theory machine“ oder, wie sie sie auch nannten, den „logic theorist“ (LT) einer größeren Öffentlichkeit vor. Der LT sollte Beweise von Theoremen in symbolischer Logik entdecken. Ähnlich wie Newell 1955 die Schachmaschine führten Newell und Simon 1956 den LT als Teil eines größeren Forschungsprogramms zum Verstehen komplexer Informationsverarbeitungssysteme allgemein ein. Der LT erscheint in dem Text entsprechend als informationsverarbeitendes System, das auch mit der Fähigkeit ausgestattet ist, Erfahrungen zu sammeln und zu lernen (Newell, Simon 1956, S. 74):

> „But, although the program of LT does not change as it accumulates experience in solving problems, learning does take place in one very important respect. The program stores the new theorems it proves, and these theorems are then available as building blocks for the proofs of subsequent theorems.“

Bei der RAND-Corporation hatten Newell und Simon in einem Projekt gearbeitet, in dem der „Organismus“ Luftverteidigungs-Frühwarnstation im Labor simuliert wurde, um dessen Verhalten zu untersuchen. Ein solches Vorgehen beschrieben sie 1956 als eine Möglichkeit, komplexe System zu untersuchen (Newell, Simon 1956, S. 61): „One tactic for exploring the domain of complex systems is to synthesize some and study their structure and behavior empirically.“ In dem Papier ist es jedoch nicht ein Labor, in dem komplexe System synthetisiert werden sollen, sondern die Rechenmaschine. Newell, und Simon gehen hier davon aus, daß sie Informationsverarbeitungssysteme auf dem Computer realisieren können. Ein „high speed digital computer“ erscheint – ohne daß dies weiter ausgeführt wird – als ein „physikalisches System“, das fast[127] jedes Informationsverarbeitungssystem realisieren kann (Newell, Simon 1956, S.62): „The high speed digital computer is a physical system that can realize almost any information processing system and our research is oriented toward using it.“ Statt komplexe Systeme wie den LT im Labor zu simulieren, sollten sie nun auf dem Computer realisiert und ihr Verhalten hier empirisch untersucht werden.

Im Labor wurde der Organismus Luftverteidigungs-Frühwarnstation dadurch synthetisiert, daß Menschen und Maschinen in einem möglichst realitätsgetreuen Setting zusammenarbeiteten. Ihr Verhalten wurde in verschiedenen simulierten Situationen – z. B. einem bestimmten Angriffsfall – beobachtet und untersucht. Auch

---

127 Ähnlich sagte die FORTRAN-Gruppe von ihrer Sprache, daß sie „virtually any numerical procedure“ ausdrücken könne. Auch von Neumann sprach in dem „First Draft“ davon, daß sein Gerät „as nearly as possible all purpose“ (2.3) sein sollte. Immer wird ein „fast“ eingeflochten, das eine Restunsicherheit der jeweiligen Autoren in bezug auf ihre Behauptung dokumentiert.

bei der Synthetisierung ihrer Informationsverarbeitungssysteme auf dem Computer wollten Newell und Simon die Realität möglichst weitgehend nachbilden: Bei der Programmierung des LT ließen sich Newell und Simon von dem aktuellen Wissen über menschliches Problemlösen leiten. So verweisen sie 1956 darauf, daß sie für den Entwurf des LT von der aktuellen Forschung zu menschlichem Problemlösen viel gelernt hätten und drücken ihren Dank aus (S. 61, Fußnote 3) „to research on human problem solving, for the basic structure of the program of LT."

Anhand des – nach dem Vorbild des menschlichen Problemlösens – hergestellten Programms wollten Newell und Simon anschließend über Informationsverarbeitungssysteme allgemein – also unter anderem Menschen – lernen. Fertiggestellte Programme wollten sie dazu empirisch untersuchen (Newell, Simon 1956, S. 79):

> „In these and other ways, the logic theorist is an instructive instance of a complex information process. We expect to learn more about such processes when we have realized the logic theorist in a computer and studied its operations empirically".

Was damit gemeint war, formulierten sie ein Jahr später auf der „Western Joint Computer Conference" deutlicher in dem Vortag mit dem Titel „Empirical Explorations with the Logic Theory Machine. A Case Study in Heuristics". Der LT, so die Autoren hier (Newell, Shaw, Simon 1957, S. 109, S. 110),

> „was devised to learn, how it is possible to solve difficult problems such as proving mathematical theorems, discovering scientific laws from data, playing chess, or understanding the meaning of human prose. [...] Our research strategy in studying complex systems is to specify them in detail, program them for digital computers, and study their behavior empirically by running them with a number of variations and under a variety of conditions. This appears at present the only adequate means to obtain a thorough understanding of their behavior."

Wie in einem Labor verschiedene Situationen simuliert werden, sollte also das Verhalten des LT (oder anderer komplexer Informationsverarbeitungssysteme) unter verschiedenen Bedingungen untersucht werden. Doch was macht Programme zu einem interessanten Objekt empirischer Forschung? Eine sinnvolle empirische Untersuchung setzt voraus, daß man im Vorhinein nicht mit Sicherheit weiß, wie sich das zu untersuchende Objekt verhalten wird – so wie das bei einer simulierten Luftverteidigungsstation auf Grund der beteiligten Menschen der Fall ist, deren Verhalten nicht prinzipiell vorhersehbar ist. Das scheint für ein Programm – besonders wenn man es selbst geschrieben hat, wie Newell und Simon den LT – zunächst nicht der Fall zu sein. Doch wie der LT als ein Fallbeispiel für Problemlösungsprozesse allgemein eingeführt wurde, schien er sich auch in einer Weise zu verhalten, wie das für menschliches Problemlösen charakteristisch ist: Vor einem Programmlauf schien unklar zu sein, wie das Programm das gegebene Problem lösen würde. Für diese Sicht auf den LT spielte das Konzept der Heuristik ein zentrale Rolle. Auf dieses gehe ich nun genauer ein.

Auf der „Western Joint Computer Conference" von 1957 hielten Newell und Shaw auch einen Vortrag, der sich mit der Programmierung des LT befaßt, „Programming the Logic Theory Machine" (Newell, Shaw 1957). Hier wird genauer deutlich, wie sie sich menschliches Problemlösen und Entscheiden – und damit das vom LT zu kopierende Vorgehen – vorstellten. Programme wie der LT müssen, so Newell und Shaw hier, wie ein Mensch in einer vergleichbaren Situation, Problemlösungsaktivität entwickeln – und das heißt genauer „ ‚problem-solving' activity, in the sense that one has no path to the solution at the start, except to apply vague rules of thumb, like ‚consider the relevant features' " (Newell, Shaw 1957, S. 230). Beim Beweisen geht der LT ihrer Darstellung nach im wesentlichen in dieser Weise vor – und damit so wie ein Mensch (Newell, Shaw 1957, S. 230):

> „A companion paper has discussed a system called the Logic Theory Machine (LT), that discovers proofs for theorems in symbolic logic in much the same way as a human does. It manipulates symbols, it tries different methods, and it modifies some of its processes in the light of experience".

Beim Beweisen und auch beim Problemlösen allgemein manipulieren Mensch wie LT demnach also Symbole und probieren auf Grundlage von Faustregeln verschiedene Verfahren aus, um zum Ziel zu gelangen. Um diese Vorstellung von Problemlösungsprozessen in die Programmierung einzuführen, bildeten Newell, Shaw und Simon den Begriff der heuristischen Programme.[128] In Abgrenzung zu „algorithmischen Programmen", die nach ihnen systematisch und immer eine Problemlösung fanden, aber nur für die Programmierung weniger komplexer Vorgänge als das menschliche Problemlösen geeignet waren, probierten heuristische Programme verschiedene programmierte Methoden aus. Dabei fanden sie zwar nicht immer eine Lösung – und wenn, unter Umständen nicht die optimale –, beanspruchten dafür aber in der Regel erheblich weniger Rechnerzeit. Heuristischer Programmierung unterlag damit die gleiche Vorstellung von Problemlösungs- und Entscheidungsprozessen, die Simon in „Administrative Behavior" bei der Untersuchung von Verwaltungen entwickelt hatte. Ein heuristisches Programm wie der LT probierte verschiedene programmierte Methoden aus; zu Beginn eines Programmlaufs schien sein Vorgehen nicht festzustehen – zumal er sich den Autoren zufolge selbst vor dem Hintergrund der gemachten Erfahrungen veränderte. So gesehen wird er zu einem interessanten Objekt empirischer Forschung. Die programmierte Maschine scheint sich schöpferisch betätigen zu können. 1958 hielten Newell und Simon fest (Newell, Simon 1958, S.8; nach Weizenbaum 1977, S. 187):

---

128 Der Begriff Heuristik bedeutete bei ihnen zunächst folgendes (Newell, Shaw, Simon 1957, S. 114, Fußnote): „As a noun, ‚heuristic' is rare and generally means the art of discovery. The adjective ‚heuristic' is defined by Webster as: serving to discover or find out." Heuristische Programme sollen so verstanden etwas entdecken wie der LT Beweise von Theoremen.

„Es gibt nunmehr in der Welt Maschinen, die denken, lernen und schöpferisch tätig sind. Darüberhinaus wächst ihre Fähigkeit auf diesen Gebieten zunehmend, bis – in absehbarer Zukunft – der Bereich von Problemen, die sie bearbeiten können, sich mit dem Bereich deckt, der bis jetzt dem menschlichen Denken allein vorbehalten war."

Der „logic theorist" ist also nach dem Vorbild menschlicher Problemlösungsprozesse erstellt und soll umgekehrt Rückschlüsse auf menschliche Informationsverarbeitung zulassen. Die Rechenmaschine erscheint als eine Art Labor, in dem der LT und andere Informationsverarbeitungssysteme synthetisiert werden können. Unter LT kann dabei zum einen das Programm und zum anderen die programmierte Maschine verstanden werden (Newell, Shaw und Simon 1957, S. 110):

„LT, of course, is a program, written for the JOHNNIAC, represented by marks on paper or holes in cards. However, we can think of LT as an actual physical machine and the operation of the program as the behavior of the machine. One can identify LT with JOHNNIAC after the latter has been loaded with the basic program, but before the input of data."

In dem besprochenen FORTRAN-Text von 1957 erschien der „translator" zum einen als Programm, zum anderen auch die IBM 704 ausgerüstet mit diesem Programm (vgl. Abschnitt 4.4.2.3.2); genauso wird der LT hier zunächst als ein Programm gesehen, das durch Schriftzeichen oder Löcher in Karten repräsentiert wird, zum anderen als der JOHNNIAC, der mit dem LT ausgerüstet ist. Die Operationen des Programms werden als Verhalten der programmierten Maschine vorgestellt. Der LT kann als operierendes Symbolsystem gedacht werden, weil für die in ihm beschriebenen Operationen die ausführende Maschine direkt mitgedacht wird. Wie für den „FORTRAN translator" wird auch für den „logic theorist" die Hybridobjekt-Sichtweise deutlich: Die Operationen des LT werden als menschenähnlich wahrgenommen, er kann als operierendes Symbolsystem und auch als programmierte Maschine gedacht werden.

4.6.1.3 IPL – die Sprache

Um die Programmierung zunächst des LT und schließlich heuristischer Programme allgemein zu ermöglichen, entwickelten Newell, Shaw und Simon eine eigene Sprache. 1956 stellten Newell und Simon die „logic language" (LL) vor, die sie als „formal language, of the nature of a pseudo-code, that is suitable for digital computers" (Newell, Simon 1956, S. 61) einführten. LL gilt heute als die erste, allerdings nie implementierte Version von IPL. Die „Information Processing Language" (IPL) bezeichneten sie 1957 als eine „Programmiersprache". Daß dieser Begriff zu der damaligen Zeit noch relativ neu und erklärungsbedürftig war, wird daran deutlich, daß sie eine Definition gaben (Newell, Shaw 1957, S. 231): „By a programming language we mean a set of symbols and conventions that allows a programmer to specify to the computer what processes he wants carried out." 1957 war ein IPL-Interpreter für den JOHNNIAC-Rechner der „RAND Corporation" implementiert und auch schon getestet worden. Ihre Sprache konnte nach Meinung der Autoren für eine für Menschen

verständliche Beschreibung von Informationsverabeitungsprozessen und auch für die Programmierung verwendet werden, worauf ich zunächst eingehe. Anschließend zeige ich auf, daß die Art und Weise, wie sie IPL darstellten, sowie in IPL realisierte Konzepte den Anspruch widerspiegeln, Informationsverarbeitungs- und Problemlösungsprozesse beschreiben und programmieren zu können.

Für die Untersuchung komplexer Informationsverarbeitungssysteme, für die der LT ein Beispiel sein sollte, unterscheiden Newell und Simon 1956 zwei Probleme: das Spezifikations- und das Realisierungsproblem. Während das Spezifikationsproblem in der Feststellung der Charakteristika eines Systems besteht, ist das Realisierungsproblem, daraus ein sich so wie das Originalsystem verhaltendes physikalisches System zu konstruieren (Newell, Simon 1956, S. 62):

> „To study all but the simplest of such systems [information processing systems], it is necessary to make a complete and precise statement of their characteristics. This statement, or specification, must be sufficiently complete to determine the behavior of the system once the initial and boundary conditions are given. [...] Once the specification has been given, a second problem is to find or construct a physical system that will behave in the manner specified."

Zwar hielten Newell und Simon zu Beginn des Papiers noch fest, daß sie nur die Spezifikation des LT, nicht jedoch die Realisierung in einem Rechner behandeln würden; da LL – die Sprache, mit der der LT in dem Papier spezifiziert wird – für Menschen jedoch recht gut verständlich sei, und auch von einem Computer interpretiert werden könne, erscheinen ihnen die Beschreibung der Charakteristika eines Systems und dessen Realisierung als nur eine Aufgabe (Newell, Simon 1956, S. 62):

> „The machine code, although suitable for communicating with the computer, is not at all suitable for human thinking or communication about complex systems. For these purposes, we need a language that is more comprehensible (to humans), but one that can still be interpreted by the computer by means of a suitable program. [...] Hence the two problems of specification and realization of an information processing system are subsumed under the single task of describing the system in an appropriate pseudo-code."

LL sollte also dazu dienen, den LT zu spezifizieren – d. h. dazu, seine Charakteristika genügend vollständig und präzise zu beschreiben, daß sein Verhalten bei Angabe der Anfangs- und Grenzbedingungen bestimmt ist (vgl. vorletztes Zitat). Gleichzeitig sollte LL auch vom Computer interpretiert werden können.[129] Daß die Charakteristika eines Informationsverarbeitungssystems tatsächlich mit einer Sprache wie LL, die auch von den Erfordernissen der Programmierung geprägt ist, beschrieben werden kann, erscheint hier als selbstverständlich. In einem 1961 erschienenen Artikel zu dem „General Problem Solver", dem Folgeprojekt des „Logic Theorist", hielten Newell und Simon in Folge fest, daß psychologische Theorien in Form von Programmen

---

129 Auch wenn LL selbst nicht implementiert wurde, sondern ein Vorläufer des später implementierten IPL war, war seine Form doch von den Erfordernissen der Maschine bestimmt (Newell, Simon 1956, S. 62): „Although this paper is concerned exclusively with the specification problem, the form of the language chosen is dictated also by the requirements of realization."

ausgedrückt werden können. Ihr Versuch, menschliche Informationsverarbeitung im Labor der RAND-Corporation formal zu beschreiben, fand so einen Abschluß (Newell, Simon 1961, S. 279):

> „This article is concerned with the psychology of human thinking. It sets forth a theory to explain how some humans try to solve simple formal problems. The research from which the theory emerged is intimately related to the field of information processing and the construction of intelligent automata, and the theory is expressed in the form of a computer program."

Da die Realisierung des Systems in der Konstruktion eines sich in der spezifizierten Weise verhaltenden physikalischen Systems besteht (vgl. drittletztes Zitat), erscheinen das Beschreiben der Charakteristika eines Systems bzw. das Ausarbeiten einer Theorie und (materielles) Konstruieren bei der Programmierung mit IPL als nur eine Aufgabe.

Entsprechend dem Anspruch, daß mit IPL (bzw. LL) Problemlösungs- und Informationsverarbeitungsprozesse beschrieben und realisiert werden sollten, war der Aufbau dieser Sprachen – wie schon der des LT – von den Theorien inspiriert, die besonders Simon über menschliches Denken hatte. Während die Sprache vor allem von Newell und Shaw erarbeitet wurde, bestand Simons Erinnerung zufolge seine Rolle im Abgleichen des Sprachdesigns mit menschlichen Denkprozessen (Simon 1991, S. 204): „My role consisted primarily of comparing proposed language designs with the analogous human functions". Die Ausrichtung der IPL-Entwicklung an der Vorstellung der Autoren von menschlichen Denkprozessen dokumentiert sich in verschiedener Weise in der Konzeption und der Funktionsweise von IPL sowie der Art und Weise, wie die Sprache dargestellt wurde:

- Wie das menschliche Denken als Informationsverarbeitung vorgestellt wurde, sollten IPL-Programme nicht nur Zahlen, sondern Informationen allgemein verarbeiten. Diese sollten im Speicher in Listenstrukturen abgelegt werden (Newell, Shaw 1957, S. 233): „Lists are the general units of information in the memory."
- Nach dem Vorbild des menschlichen Gedächtnisses sollte die IPL-Speicherorganisation assoziativ sein und dadurch Lernen ermöglichen:[130] „Each symbol should lead to other symbols that are linked to it, and these links should be acquired by learning." (Simon 1991, S. 204) Daß nicht erst Simon in seiner Biographie von 1991 den Begriff der Assoziation benutzte, zeigt folgende Textpassage (Newell, Tonge 1960, S. 205): „The relationships between elements of data are restructured during the program's operation. New associations must be represented and old ones deleted."
- Die IPL-Speicherorganisation erlaubte die automatische Zuteilung von Speicherplatz zur Laufzeit eines Programms, wohingegen z. B. bei FORTRAN der gebrauchte Speicherplatz schon vor dem Programmablauf festgelegt werden mußte.

---

130 Der LT wurde entsprechend als Programm gesehen, das lernen kann (s. letzter Abschnitt).

Während eines Programmlaufs konnten so Listenstrukturen entstehen, die der Programmierer nicht schon im Vorhinein angekündigt hatte. Listen und die mit IPL realisierte Speicherorganisation spiegelten so den Anspruch wider, daß Problemlösungsaktivität programmiert werden sollte, für die zu Beginn der zu beschreitende Lösungsweg noch nicht feststeht.

- 1960 erscheint IPL in einem Artikel von Newell und Tonge als ein Mittel, das einem Programmierer erlaubt, der Rechenmaschine sein nur unvollständiges Wissen über das zu programmierende Problem mitzuteilen (S. 205):

  „There exist many tasks that men can perform reasonably well without knowing in detail how to perform them. [...] Getting the computer to play chess or prove theorems, using the same problem-solving techniques as humans, poses communication problems with the machine far beyond those of expressing formal algebraic manipulations. The user must somehow communicate to the machine his incomplete knowledge of how to behave in these complex situations. The IPL series of programming languages has been developed as an aid in constructing problem-solving programs using the adaptive, cut-and-try methods (‚heuristics') characteristic of human behavior – as a research tool in the study of heuristic problem solving."

  IPL erscheint hier als Sprache, die dabei hilft, der Maschine zu kommunizieren, wie ein Mensch Probleme löst, obwohl dies dem programmierenden Menschen selbst nur teilweise bewußt ist. Ein in IPL erstelltes Programm scheint in ähnlicher oder gleicher Weise Probleme zu lösen, wie ein Mensch.
- Entsprechend dem Anspruch, mit IPL Denkvorgänge zu programmieren, deren Ablauf im Vorhinein nicht vollständig kommuniziert werden kann, sollte die Sprache erlauben, flexibel mit Speicherplatz und der Spezifikation von Prozessen umzugehen. Der IPL-Computer erscheint so als flexibles Gerät, dem so etwas wie eigenständiges Problemlösen eher zugetraut werden kann (Newell, Shaw 1957, S. 231):

  „1) There should be no restriction in the number of different lists of items of information to be stored. [...] 2) There should be no restriction in the nature of the items in a list. [...] 3) It should be possible to add, delete, insert, and rearrange items of information in a list at any time and in any way. [...] 4) It should be possible for the same item to appear on any number of lists simultaneously."

IPL wurde also als eine Programmiersprache konzipiert, die erlauben sollte, die Charakteristika von Informationsverarbeitungssystemen zu beschreiben, auch wenn der genaue Ablauf des zu programmierenden Prozesses nicht vollständig kommuniziert werden konnte. Entsprechend sollte sie ermöglichen, daß die Rechenmaschine mit Informationen allgemein arbeiten, assoziativ lernen und eigenständig Probleme lösen kann. Im Zentrum der IPL-Entwicklung stand die flexible Organisation des maschinellen Gedächtnisses: Nach dem Vorbild des menschlichen Gedächtnisses sollten Informationen in Listen abgelegt und während eines Programmlaufs variabel manipuliert und kombiniert werden können. Die Verwaltung des Speicherplatzes wurde mehr noch als in FORTRAN in die Maschine hineinverlegt: Eine als „NSS-

memory"[131] bekannt gewordene Speicherorganisation erlaubte, Speicherplatz zur Laufzeit flexibel zu vergeben. Dazu wurde der gesamte freie Speicherplatz vor einem Programmlauf zu einer Liste zusammengefaßt. Wenn Speicherplatz zur Laufzeit benötigt wurde, wurde er vom Anfang dieser Liste abgehängt; umgekehrt wurde nicht mehr gebrauchter Speicherplatz an den Anfang dieser Liste zurückgegeben.

Obwohl LL – anders als Maschinencode – auch für den Menschen gut verständlich sein sollte, erinnern IPL-Programme ihren äußeren Form nach an Assemblerprogramme (Sammet 1969, S. 389): „IPL-V is really the machine language for a hypothetical computer which does list processing." Die Sprache soll jedoch, wie Sammet (1969, S. 390) in Anlehnung an ein von Newell und anderen 1965 verfaßtes Handbuch zu IPL festhält, zu Beginn der 60er Jahre noch als problemorientierte Sprache empfunden worden sein – was deshalb nicht unwahrscheinlich ist, weil zu dieser Zeit Assemblerprogrammierung noch der Normalfall war.[132] IPL bestand aus einer langen Liste durchnumerierter Informationsprozesse J1, J2, J3 usw., die Listenmanipulationen wie das Kopieren, Einfügen oder Löschen einer gesamten Liste oder eines Listenelements ermöglichten.[133] 1963 stellte IPL bereits 200 verschiedene Prozesse zur Verfügung. Das wurde als Service für Benutzer verstanden und sollte ermöglichen, daß in der Listenverarbeitung neue Personen nicht selbst alle Einzelheiten herausfinden mußten (Newell 1963, S. 88):

> „[...] since list processing was relatively new, people should not have to discover for themselves how to put the little pieces together. Hence, a substantial array of basic processes should be provided in the language as a means of (automatically) communicating to the user the things he could do with the language. This argument [...] was accepted an led to the 200 odd primitives that now exist in IPL."

Mit IPL V wurde die Entwicklung von IPL Mitte der 60er Jahre eingestellt, da LISP IPL als Programmiersprache für Listenverarbeitung ablöste.

---

131 NSS steht für Newell, Shaw, Simon.

132 Daß zu dieser Zeit die assemblerähnliche Notation von IPL von den meisten Programmierern nicht als problematisch empfunden wurde, wird an folgendem Zitat deutlich. Gelernter, selbst Leiter der FLPL-Entwicklung, schreibt hier zum Vergleich von IPL und der ebenfalls listenverarbeitenden FORTRAN-Erweiterung FLPL (Gelernter et al. 1960, S. 100): „Comparision of the two languages in terms of programming convenience is largely taste-dependent. Again, the authors feel that an algebraic compounding of mnemonically-named expressions is preferable to a linear sequencing of ‚catalog-number' designed routines, but programmers experienced in the use of IPL seem to feel little advantage in the change of programming format. A new programmer [...] is likely to require the same amount of time to gain proficiency in either language."

133 Sammet (1969, S. 388ff.) gibt eine Einführung in das Programmieren mit IPL.

4.6.1.4 Zusammenfassung: Die Rechenmaschine als behavioristisches Labor

Bei ihrer Arbeit mit der Rechenmaschine ging es den IPL-Entwicklern Newell, Shaw und Simon weniger um das Erstellen von Programmen, die bestimmte Arbeiten wie das Rechnen oder Kodieren übernehmen sollten. Vielmehr war der Computer für sie – wie in den letzten Abschnitten deutlich wurde – ein Forschungsmittel bei der Frage danach, wie Menschen Informationen verarbeiten und Probleme lösen. Dazu wollten sie menschliche Problemlösungsprozesse in IPL spezifizieren, programmieren und auf dem Computer empirisch untersuchen. Die erstellten Programme stellten für sie Theorien über menschliches Denken dar. Mit IPL drangen Programmiersprachen und der Computer somit in die Geistes- und Sozialwissenschaften ein – und zwar nicht als Hilfsmittel bei der Datenverarbeitung, sondern als Mittel der Forschung und Ausdrucksmittel für Theorien. In einem Artikel von 1958 formulierten sie das wie folgt (Shaw, Newell, Simon, Ellis 1958, S. 119):

> „[...] humans carry out information processing of a complexity that is truly baffling. Given the urge to understand either how humans do it, or alternatively, what kinds of mechanisms might accomplish the same tasks, the computer is turned to as a basic research tool."

Das Zitat macht auch deutlich, daß für Newells, Shaws und Simons Forschungen zu menschlicher Informationsverarbeitung ein behavioristischer Handlungsbegriff zentral war. Die Frage danach, wie Menschen Informationen verarbeiten, ersetzen sie durch die Frage, welche Mechanismen dieselben Aufgaben ausführen. Daß Menschen Informationen in einer Weise verarbeiten könnten, wie sie von keinem Mechanismus nachvollzogen werden kann, scheinen die Autoren nicht in Erwägung zu ziehen. Aus einer behavioristischen Perspektive, aus der Handeln mit dem Ablauf von Reiz-Reaktionsabläufen mechanistisch erklärt werden kann, ist das eine folgerichtige Annahme.

Auch die Vorstellung von Entscheiden, die in der Konzeption heuristischer Programme zum Ausdruck kommt, verweist auf einen behavioristischen Hintergrund. Schon 1945 hatte Simon Entscheiden als Auswahl unter einer gegebenen Menge von Möglichkeiten konzipiert, wobei eine rationale Entscheidung zur Erreichung eines gegebenen Ziels diente. Auch heuristische Programme wählen – den Autoren zufolge wie ein Mensch auch – bei einem gegebenen Ziel unter einer gegebenen Menge programmierter Verfahren nach gegebenen Kriterien aus. Der „General Problem Solver" (GPS) z. B., das Folgeprojekt des „Logic Theorist", arbeitete auf Grundlage einer solchen Heuristik, der sogenannten Mittel-Ziel-Analyse. Nach dieser wird für jedes Problem die Differenz zwischen dem augenblicklichen und dem gewünschten Zustand festgestellt (z. B. Kind zu Hause, soll aber in den Kindergarten). Anschließend werden die Operatoren ausgewählt, die den augenblicklichen in den gewünschten Zustand überführen (z. B. das Auto, mit dem das Kind in den Kindergarten gefahren wird) (vgl. Dreyfus 1985, S. 26f.). Die Frage danach, wie das gegebene Ziel entsteht

und aus welchem Grund es als anstrebenswert empfunden wird, und damit die Frage nach dem Sinn des Handelns ist in ihren Überlegungen kein Thema – ähnlich wie das in den in Kapitel 2 und 3 besprochenen Texten der Fall war (vgl. besonders Abschnitt 3.1.1, 3.3), und wie es für behavioristische Erklärungsansätze kennzeichnend ist..

Die Forschungsstrategie, nach der Newell, Shaw und Simon vorgingen – komplexe Systeme zu synthetisieren und anschließend ihr Verhalten unter verschiedenen Bedingungen empirisch zu untersuchen – kann als Fortsetzung ihrer Arbeit in den Laboren der RAND-Corporation gesehen werden: Hier wurde in einem behavioristischen Versuchssetting ein komplexes Mensch-Maschine-System – eine auch als „Organismus" betrachtete Luftverteidigungs-Frühwarnstation – simuliert und ihr Verhalten unter verschiedenen Testbedingungen untersucht. Für Newell und Simon wurde der Computer nach Abschluß dieser Arbeiten zum behavioristischen Labor: Mit Hilfe von LL bzw. IPL wollten sie hier komplexe informationsverarbeitende Systeme synthetisieren und anschließend ihr Verhalten empirisch untersuchen. Die Komponenten, aus denen sie ihre Systeme zusammensetzten, waren nun nicht mehr die Menschen und Maschinen der Laborsimulation, sondern die „basic processes" oder auch „information processes", die in IPL zur Verfügung gestellt wurden und aus denen ein IPL-Programm bestand.

Dieses Vorgehen erinnert an das von Neumannsche „axiomatischen Verfahren" – entsprechend verwiesen Newell und Simon 1956 (S. 279) darauf, daß ihre Arbeit in enger Beziehung zu dem Forschungsgebiet der Konstruktion intelligenter Automaten steht (s. o. Abschnitt 4.6.1.3). Von Neumann konzipierte Automaten als Aggregate sehr vieler biologischer oder technischer Elemente, von denen er annahm, sie als Stimulus-Response-Automatismen beschreiben zu können. Das Verhalten von Automaten kann so gesehen durch das Zusammenwirken des Verhaltens dieser Automatismen erklärt werden. Der behavioristisch-atomistischer Blick erlaubte, Automaten als Symbolsysteme zu konzipieren (vgl. Kapitel 3). Newell, Shaw und Simon gingen ähnlich vor, fingen jedoch gewissermaßen auf der anderen Seite an: bei den Symbolsystemen. Sie beschrieben – spezifizierten – Informationsverarbeitungssysteme mit Hilfe einer Sprache, die aus „Informationsprozessen" bestand; sie betrachteten Informationsverarbeitungssysteme also als realisiert durch elementare Prozesse, die auch als Ein-/Ausgabeautomatismen betrachtet werden können.[134] Diese Beschreibung konnte ihnen dann zur physikalischen Realisierung des Informationsverarbeitungssystems auf dem Computer dienen.

Ähnlich wie Automaten können Informationsverarbeitungssysteme bei Newell, Shaw und Simon als Symbolsysteme, als Maschinen oder als Menschen gedacht

---

134 So sind Informationsprozesse als mathematische Funktion beschreibbar (Newell, Simon 1956, S. 62): „The IP's [information processes] are, mathematically speaking, functions from the input memories and their contents to the symbols in the output memories."

werden.[135] Zwar sprachen Newell, Shaw und Simon nicht mehr von Neuronen oder dem physikalischen Aufbau des Nervensystems – ein behavioristisch-atomistischer Blick auf Informationsverarbeitungssysteme lag jedoch auch ihren Überlegungen zugrunde: Informationen schienen auf Grundlage elementarer und deterministischer Informationsprozesse verarbeitet zu werden; Informationsverarbeitung und Problemlösungsprozesse schienen ohne Bezug auf Fragen des Sinns erklärbar zu sein. Die Grenzen zwischen natürlich und technisch sowie zwischen Organismus und Symbolsystem werden auch in ihren Arbeiten undeutlich.

4.6.2 LISP

LISP wurde ab Ende 1958 am MIT unter der Leitung von John McCarthy entwickelt. Mitte der 60er Jahre löste es IPL als Hauptsprache der „Artificial Intelligence" ab. Da auch für die LISP-Entwicklung ein konkretes Nutzungsvorhaben den zentralen Ausgangspunkt bildete – die Realisierung des „Advice Takers" –, betrachte ich nach einer Schilderung der Vorgeschichte von LISP zunächst dieses Vorhaben und bespreche dazu McCarthys Papier „Programs with Common Sense". Deutlich wird hier, daß McCarthy LISP mit der Vorstellung entwickelte, die Sprache für die Programmierung und auch als Ein-/Ausgabesprache für die Kommunikation zwischen Mensch und programmierter Maschine zu verwenden. Wie in der menschlichen Kommunikation beim Instruieren eines Menschen sollten imperative und deklarative Sätze möglich sein. Nach der Besprechung des „Advice Taker"-Projektes gehe ich auf die LISP-Sprache näher ein. Die wichtigste Grundlage meiner Betrachtung ist dabei das erste Papier, in dem McCarthy LISP 1960 einer größeren Öffentlichkeit vorstellte, „Recursive Functions of Symbolic Expressions". Die Sprache wird hier aus rein mathematisch-formalen Erwägungen abgeleitet und erscheint als Formalismus für die Berechenbarkeitstheorie bzw. – wie McCarthy es ausdrückte – für eine Theorie des Berechnens; die Rechenmaschine spielt für die Gestalt der Sprache scheinbar keine Rolle. Gleichzeitig soll LISP dem Papier zufolge dazu dienen, gesunden Menschenverstand zu programmieren. Deutlich wird – wie ich in der abschließenden Zusammenfassung argumentiere – ein Orientierungsrahmen, vor dessen Hintergrund menschliches Denken und Berechnen bzw. logisches Schlußfolgern im wesentlichen dasselbe zu sein scheinen.

---

135 1972 formulierten sie das prägnant wie folgt (Newell, Simon 1972, S. 870): „The basic point of view inhabiting our work has been that programmed computer and human problem solver are both species belonging to the genus IPS [information processing system]."

4.6.2.1 Vorgeschichte

John McCarthy (geb. 1927) studierte Mathematik am California Institute of Technology und an der Universität Princeton, wo er 1951 seinen PhD erhielt und anschließend noch zwei weitere Jahre als Lehrkraft arbeitete. In Princeton lernte er auch Marvin Minsky (geb. 1927) kennen, der dort ebenfalls Mathematik studierte und 1954 seinen PhD erwarb. 1948 besuchte er das Hixon Symposium, auf dem John von Neumann seinen Vortrag über Automaten[136] hielt. Der auf der Konferenz thematisierte Vergleich zwischen Rechenmaschinen und dem Gehirn war für ihn, so McCarthy in einem Interview, das ich im Dezember 1994 mit ihm führte, der Ausgangspunkt für sein Interesse an dem, was er später „Artificial Intelligence" (AI) nennen sollte. Während seiner Studienzeit in Princeton sprach er über seine Ideen zu Künstlicher Intelligenz mit John von Neumann, der ihm riet, diese aufzuschreiben; nach näherem Nachdenken gab McCarthy dieses Vorhaben jedoch zunächst wieder auf.

Den Sommer 1952 verbrachte McCarthy in Claude Shannons Labor an den „Bell Telephone Laboratories". Zusammen mit Shannon begann er hier, Beiträge für einen Band zu sammeln, der sich nach McCarthys Wunsch eigentlich mit künstlicher Intelligenz beschäftigen sollte, aber auf Grund der Thematik der schließlich eingereichten Beiträge von ihm und Shannon „Automata Studies" genannt wurde. Das Buch erschien 1956. Shannon und McCarthy schrieben zu diesem Band das Vorwort. Hier stellten sie – ähnlich wie von Neumann – fest, daß das Gehirn als endlicher Automat betrachtet werden kann (Shannon, McCarthy 1956, S. vi): „Indeed, it should be remembered that essentially all actual physical machines and even the brain itself are, or can be reasonably idealized to be, of this form." Zudem stellten sie heraus, daß das Verstehen und der Nachbau – Analyse und Synthese – des Gehirns zu den herausforderndsten und schwierigsten Problemen der Zeit gehörten (Shannon, McCarthy 1956, S. v):

> „Among the most challenging scientific questions of our time are the corresponding analytic and synthetic problems: How does the brain function? Can we design a machine which will simulate a brain?"

Daß diese Fragen bald beantwortet würden, war ihrer Meinung nach mehr als wahrscheinlich (ebd.): „Recent progress in various fields leads to an optimistic view toward the eventual and not too remote solution of the analytic and synthetic problems."

Nachdem McCarthy von 1951 bis 1953 als „Acting Assistant Professor" an der Universität Stanford gearbeitet hatte, ging er als „Assistant Professor of Mathematics" an das Dartmouth College. IBM stiftete zu dieser Zeit einen Rechner für das MIT und die Colleges und Universitäten Neu Englands, der am MIT aufgestellt werden sollte. In diesem Zusammenhang unternahm Nathaniel Rochester, „Manager of Information Research" bei IBM in Poughkeepsie, New York, eine Besuchsreise bei den Colleges

---

136 Diesen Vortrag bespreche ich ausführlich in Abschnitt 3.1.1.

und Universitäten Neu Englands. Rochesters Reise diente wohl zum Prüfen, Festlegen und Organisieren der von den einzelnen Universitäten benötigten Rechnerzeit. Ein Drittel der Rechenzeit der Maschine, die rund um die Uhr lief, ging an das MIT, ein Drittel an die anderen Universitäten bzw. Colleges und ein Drittel an IBM selbst. McCarthy wurde in der Rechnernutzergruppe der Repräsentant von Dartmouth, was wohl der Grund dafür war, daß Rochester ihm für den Sommer 1955 einen Job bei IBM anbot. Im Rahmen dieser Arbeit hatte McCarthy das erste Mal Gelegenheit, einen Computer zu programmieren.

Im Sommer 1956 organisierte McCarthy das „Dartmouth Summer Research Project on Artificial Intelligence" mit finanzieller Unterstützung der „Rockefeller Foundation". Er überzeugte Shannon, Marvin Minsky[137] und Rochester, mit ihm das Organisationskomitee zu bilden. Anwesend waren auf der Konferenz auch Simon und Newell. Auch von Neumann wäre gerne gekommen, wie McCarthy mir in dem Interview erzählte, war jedoch auf Grund seines Gesundheitszustandes – er starb im Februar 1957 - nicht mehr dazu in der Lage. Von dem Treffen wurde auch in der amerikanischen Tagespresse, so in „The New Yorker" kurz berichtet. Es wird heute allgemein als Gründungszeitpunkt der AI angesehen, auf dem die grundlegenden Annahmen der Disziplin formuliert wurden (Fleck 1982, S. 177): „It was at that meeting that the broad outlines of a distinctively AI approach, indeed what might be called a proto-paradigmatic structure, emerged." Die Wortschöpfung „Artificial Intelligence" wird übrigens McCarthy zugerechnet.

Auf dem „Dartmouth Summer Research Project" stellten Newell und Simon ihren „Logic Theorist" und in Zusammenhang damit auch IPL vor. Während die Idee der Listenverarbeitung und die Art ihrer Realisierung McCarthy gut gefiel, war er von der äußeren Form der Sprache weniger begeistert. Er hatte bereits von FORTRAN und der Idee, Programme in quasi-mathematischer Notation zu schreiben, gehört. Rückblickend schrieb er (McCarthy 1981, S. 174): „There was little temptation to copy IPL, because its form was based on a JOHNNIAC loader that happened to be available to them, and because the FORTRAN idea of writing programs algebraically was attractive".

Ende 1956 wurde bei IBM – angeregt durch einen Vorschlag Minskys, ein Programm zum Beweisen von Sätzen der zweidimensionalen Geometrie zu schreiben – ein entsprechendes Projekt initiiert, das wesentlich von Herbert Gelernter, der nach seiner Promotion seit Herbst 1956 bei IBM arbeitete, getragen wurde. Da das Programm wie der LT auf Grundlage von Listen arbeiten sollte, wurde zunächst in Betracht gezogen, für die Programmierung IPL, das damals nur für den JOHNNIAC

---

137 Minsky hatte zu dieser Zeit bereits eine Maschine gebaut, die ein Nervennetz kopieren und lernen simulieren sollte. Seine PhD-thesis hatte er über „Neural Nets and the Brain Model Problem" geschrieben.

verfügbar war, auf die IBM 704 zu übertragen. Auf Empfehlung von McCarthy, der neben Minsky in dem Projekt als Berater fungierte, wurde jedoch beschlossen, statt dessen FORTRAN zur Listenverarbeitung auszubauen – eine Entscheidung, die sich für ein IBM-Projekt natürlich anbot. Zudem wurde eine FORTRAN-basierte Sprache kompiliert, was zu erheblich schnelleren Laufzeiten führte, als das bei einer interpretierten Sprache wie IPL der Fall gewesen wäre (vgl. Gelernter et al. 1960, S. 219).

Die Beratertätigkeit bei der Entwicklung der FORTRAN-Erweiterung FLPL (**F**ortran **L**ist **P**rocessing **L**anguage) bot McCarthy die Möglichkeit, Erfahrungen bei der Konzeption listenverarbeitender Sprachen zu machen, eigene Ideen auszuprobieren und neue Anregungen zu sammeln. Zudem wurde FLPL für dieselbe Maschine, die IBM 704, entwickelt wie später die erste Version von LISP. Es überrascht so nicht, daß FLPL und LISP eine ganze Reihe Gemeinsamkeiten aufwiesen. So stimmen die Art, wie Listenstrukturen im Speicher der IBM 704 dargestellt werden sowie wesentliche Grundfunktionen wie besonders „car", „cdr" und „cons" weitgehend überein. Für die Zeit um 1959 herum stellt Stoyan fest (Stoyan 1980, S. 173): „Sie [LISP] entsprach im Grunde immer noch FORTRAN, nur konnte man bedingte Ausdrücke verwenden und ebenso die Listenverarbeitungsfunktionen und rekursive Funktionen definieren."

1957 erhielt McCarthy ein einjähriges Stipendium von der SLOAN-Stiftung. Er konnte deshalb frei wählen, wo er arbeiten wollte und ging zum MIT, weil dort der IBM-Rechner, auf dem auch das Dartmouth College arbeitete, stand. Hier begann er, ein Schachprogramm zu schreiben und entwickelte erste Ideen zum „time-sharing". Das MIT bot ihm in dieser Zeit eine Stelle als „Assistant Professor" am „Research Laboratory of Electronics" an, die Ende 1958 beginnen sollte. Den Sommer 1958 verbrachte McCarthy auf Einladung von Rochester am IBM-Forschungszentrum in Poughkeepsie. Hier benutzte er das Problem der Differentiation von algebraischen Ausdrücken als Beispiel, um seine Idee zu einer listenverarbeitenden, algebraischen Programmiersprache auszuarbeiten. Ende September 1958 schrieb er seine Ideen für das spätere LISP in einem ersten Memo nieder, auf das ich in Abschnitt 4.6.2.3 noch eingehe.

Die Implementation der in dem ersten Memo skizzierten, später LISP genannten Sprache begann mit dem „Artificial Intelligence Project" von Minsky und McCarthy Ende 1958 am MIT.[138] Dabei waren die dem Projekt zugeteilten „graduate students" entscheidend beteiligt. Eine stringente Planung für die Implementation gab es nicht, und jeder der einzelnen Studenten ging wesentlich dem eigenen Projekt, seiner Doktorarbeit, nach. Für LISP 1 wurden vor allem Einzelleistungen, die auf eigener

---

138 Den Hergang der Implementation schildert ausführlich (Stoyan 1980, S. 157ff.), aus dem auch die folgenden Details entnommen sind.

Initiative beruhten, wichtig. Paul Abrahams, einer dieser Studenten beschreibt die Arbeitsatmosphäre als eher spielerisch (Interview Abrahams):[139]

> „The atmosphere was playful and quirky. The best example I can think of was one afternoon when Minsky came by. ‚Eli's Surplus is having a sail of weather balloons!' he exclaimed. ‚Let's go get some, strap them on ourselves (to neutralize our weight), and go jumping!' There ensued a long discussion about where to get hydrogen, how dangerous it was, wether hot air might be a better alternative, how to heat the air etc. (But we never actually did it.)"

Wie Newell, Shaw und Simon IPL für die Realisierung ihres „Logic Theorist" entwickelt hatten, so stand auch am Anfang der LISP-Entwicklung ein konkretes Nutzungsvorhaben, der „Advice Taker". Ich werde nun zunächst auf dieses Vorhaben genauer eingehen und anschließend die sich daraus ergebende Sprache, das spätere LISP, näher beschreiben.

4.6.2.2 Der „Advice Taker"

1960 schrieb McCarthy (McCarthy 1960, S. 203): „A programming system called LISP (for LISt Processor) has been developed [...] to facilitate experiments with a proposed system called the Advice Taker." Den „Advice Taker" hatte McCarthy im November 1958 auf dem „Symposium on the Mechanization of Thought Processes" in dem Vortrag „A Program with Common Sense" vorgestellt. Der „Advice Taker" sollte nicht mehr nur – wie es noch das Anliegen eines Artikels von 1956 gewesen war (McCarthy 1956) – wohldefinierte Probleme aus dem wissenschaftlichen Bereich, sondern Probleme allgemein lösen. Es sollte gesunden Menschenverstand (engl. common sense) besitzen, worunter McCarthy die Fähigkeit verstand, in einer gegebenen Situation eine Reihe von logischen Schlüssen zu ziehen (McCarthy 1958, S. 403):

> „One will be able to assume that the advice taker will have available to it a fairly wide class of immediate logical consequences of anything it is told and its previous knowledge. This property is expected to have much in common with what makes us describe certain humans as having common sense. We shall therefore say that a program has common sense if it automatically deduces for itself a sufficiently wide class of immediate consequences of anything it is told and what it already knows."

Die für eine Problemlösung jeweils nötigen Prämissen sollen dabei weitgehend bereits im Maschinenspeicher vorhanden sein (McCarthy 1958, S. 408): „[...] all the premises can reasonably be expected to be specifically present in the memory of a machine which has competence of human order in finding its way around." Deutlich wird die Annahme, daß im Grunde alle für die Ausübung des gesunden Menschenverstandes nötigen Fakten der Maschine mit Hilfe von Prädikaten und Schlußregeln beschrieben werden können. Ähnlich sollen beliebige Verhaltensformen in Form von Automaten

---

139 Dieses Interview führte ich mit Paul Abrahams am 19. 9. 1994 in Deerfield, Massachusetts.

bzw. Programmen repräsentiert werden können (McCarthy 1958, S. 404): „All behaviors must be representable in the system. Therefore the system should either be able to construct arbitrary automata or to program in some general purpose language."

Wie das letzte Zitat deutlich macht, soll der „Advice Taker" auch in der Lage sein, selbst zu programmieren. Das entspricht dem letztendlichen Ziel des „Advice Taker"-Projektes, Programme schreiben zu können, die so effektiv lernen wie ein Mensch (McCarthy 1958, S. 404): „Our ultimate objective is to make programs that learn from their experience as effectively as humans do." Dabei betont McCarthy die prinzipielle Möglichkeit, eine Maschine dadurch zum Problemlösen bzw. Lernen zu befähigen, daß sie alle möglichen Verhaltensweisen – repräsentiert z. B. als Turingmaschinen – durchprobiert (McCarthy 1958, S. 404):

> „There is one known way of making a machine capable of learning arbitrary behaviour, and thus to anticipate every kind of behaviour: This is to make it possible for the machine to simulate arbitrary behaviour and to try them out."

Diese Möglichkeit verwirft er jedoch – ähnlich wie Newell, Shaw und Simon algorithmische Verfahren für die Programmierung komplexer Problemgebiete – als ineffizient, denn „the density of interesting behaviour is incredibly low." (McCarthy 1958, S. 404)

So sehr die Maschine beim Durchprobieren von Möglichkeiten zu ungelenk und willkürlich zu funktionieren scheint, so wenig sollen ihr bestimmte Problemlösungen aufdrängt werden. Schon 1956 hält McCarthy fest (McCarthy 1956, S. 181): „In determining a procedure which has action schemes, theories etc., we are imposing our ideas of the stucture of the problem on the machine." Statt dessen soll das Programm so gestaltet werden, daß die Maschine selbst die Probleme lösen kann und darüber hinaus in der Lage ist, sich selbst zu verbessern – eine Eigenständigkeit, wie sie ähnlich auch Newell, Shaw und Simon für den LT im Sinn hatten (McCarthy 1956, S. 181): „The point is to do this so that the procedures we offer are general (will eventually solve every solvable problem) and also are improvable by the methods built into the machine."

Auch in dem „Advice Taker"-Papier von 1958 wird McCarthys Unzufriedenheit mit Problemlösungsverfahren, die die Maschine im Vorhinein festlegen, deutlich. Solche Verfahren spiegeln seiner Meinung nach nicht die Art und Weise wieder, in der Menschen instruiert werden, sondern nur die, in der Maschinen (bisher) angewiesen wurden (McCarthy 1958, S. 405):

> „We base ourselves on the idea that in order for a program to be capable of learning something it must first be capable of being told it. [...] The main distinction between the way one programs a computer and modifies the program and the way one instructs a human or will instruct the advice taker is this: A machine is instructed mainly in the form of a sequence of imperative sentences, while a human is instructed mainly in declarative sentences describing the situation in which action is required together with a few imperatives that say what is wanted."

Wie das Ziel des „Advice Taker"-Projektes ist, daß dieser möglichst so effektiv lernt wie ein Mensch, so soll der „Advice Taker" auch instruiert werden wie ein Mensch bzw. so, wie McCarthy sich das Instruieren eines Menschen vorstellt: mit Hilfe einer Mischung aus deklarativen, also beschreibenden Sätzen und einigen Befehlen, die ihm sagen, was von ihm gewollt wird, so daß der „Advice Taker" die entsprechenden logischen Konsequenzen ableiten kann. Im Umgang mit dem „Advice Taker" soll eine Art Gespräch stattfinden, wobei „Gespräch" relativ wörtlich genommen werden kann. So soll ein Mensch dem „Advice Taker" Mitteilungen machen können, ohne vorher viel über das Programm wissen zu müssen (McCarthy 1958, S. 403):

> „The main advantages we expect the Advice Taker to have is that its behaviour will be improvable merely by making statements to it, telling it about its symbolic environment and what is wanted from it. To make these statements will require little if any knowledge of the program or the previous knowledge of the advice taker."

Umgekehrt sollte die Maschine in der Lage sein, verbal zu argumentieren, um Abstraktionen ausdrücken zu können (McCarthy 1958, S. 406): „The only way we know of expressing abstractions [...] is in language. That is why we have decided to program a system which reasons verbally."

Zentrales Anliegen des „Advice Taker"-Projektes ist, daß der „Advice Taker" in derselben formalen Sprache geschrieben wird wie der, in der die Ein- und Ausgaben getätigt und der Problembereich dargestellt werden soll (McCarthy 1958, S. 403):

> „The ‚Advice Taker' is a proposed program for solving problems by manipulating sentences in formal languages. The main difference between it and other programs such as the Logic Theory Machine and the Geometry Program of Gelernter, is that in the previous programs the formal system was the subject matter but the heuristics were all embodied in the program. In this program the procedures will be described as much as possible in the language itself and, in particular, the heuristics are all so described."

McCarthy setzt sich hier also von den IPL-Autoren und dem FLPL-Projekt dadurch ab, daß er die Problemlösungsstrategien soweit möglich in derselben formalen Sprache programmieren will, in der auch die Ein- und Ausgabe, die Gespräche mit dem „Advice Taker", stattfinden sollen. Das soll ermöglichen, daß der „Advice Taker" – der „Ratschlag Nehmer" – sich dadurch verbessern kann, daß man ihm Mittelungen über seine symbolische Umgebung und das, was man von ihm will, macht (s. o.). Wie diese Sprache aussehen soll, erläutert McCarthy an Hand eines Beispiels (McCarthy 1958, S. 407): „Assume that I am seated at my desk at home and I wish to go to the airport. My car is at my home also. The solution of the problem is to walk to the car and drive the car to the airport." Um dem „Advice Taker" eine Lösung des Problems zu ermöglichen, müssen ihm nach McCarthy zunächst die entsprechenden Prämissen eingegeben werden, die dann, formuliert als logische Prädikate bzw. Schlußregeln wie folgt aussehen: „at (I, desk)", „at (desk, home)", „at (car, home)" oder „did (go (x, y, z)) -> at (I, y)". Die Durchführbarkeit des Vorhabens, im Bereich von „x" gehend oder fahrend etwas erreichen zu können, wird als „walkable (x)" oder „drivable (x)"

dargestellt, z. B. der Wunsch, daß „ich“ zum Flughafen möchte durch „want (at (I, airport))“.

4.6.2.3 LISP – die Sprache

Ende September 1958 hatte McCarthy – nicht lange nach dem „Advice Taker“-Papier – ein erstes Memo verfaßt, in dem seine Vorstellungen für eine „algebraische Sprache für die Manipulation symbolischer Ausdrücke“, wie es im Titel heißt , skizziert sind. Die Sprache soll, so McCarthy hier, der Manipulation von Sätzen in formalen Sprachen, wie es für das „Advice Taker“-Projekt zentral war, aber auch „der Vereinfachung und Differentiation algebraischer Ausdrücke und Compilerprogrammierung für die Sprache selbst“ dienen. Zudem betont McCarthy „die Eignung der Sprache für die Formulierung heuristischer Programme, weil sie sehr geeignet sei, ‚Bäume alternativer Aktionen‘ darzustellen.“ Die in dem Memo beschriebene Sprache soll wie die des „Advice Taker“-Papiers imperative und deklarative Elemente mischen: Ähnlich wie in FORTRAN soll die Zuweisung die wichtigste Anweisung darstellen; um in dem „Advice Taker“-Papier erwähnte „property lists“ realisieren zu können, sollen mit Hilfe der Anweisung „I declare (dcl_1, dcl_2, ..., dcl_n)“ solche Listen vereinbart werden können, die Eigenschaften der Objekte beinhalten, denen sie beigestellt sind (alle Zitate zu dem Memo aus Stoyan 1980, S. 166f.).

Im April 1960 veröffentlichte McCarthy das erste Papier, das LISP einem größeren Personenkreis bekanntmachte, „Recursive Functions and Symbolic Expressions“. LISP, wird hier als ein „Programmiersystem“ eingeführt, von dem die LISP-Programmiersprache nur ein Teil ist, und zu dem unter anderem auch ein Interpreter, Lese- und Schreibvorrichtungen, Fehlererkennungshifsmittel und ein Handbuch zählen (vgl. McCarthy 1960, S. 212f.). Deutlich wird, daß LISP für eine spezielle Maschine, die IBM 704, sowie besonders für eine Anwendung, die Realisierung des „Advice Takers“ gedacht ist. Das System soll entsprechend dazu dienen, formale imperative und deklarative Sätze so bearbeiten zu können, daß der „Advice Taker“ selbständig Schlußfolgerungen ziehen und gesunden Menschenverstand ausüben kann. Mit dem Programmiersystem sollen also Programme erstellt werden können, die deklarative und imperative Sätze bearbeiten können. Gleichzeitig soll LISP auch selbst aus deklarativen und imperativen Sätze bestehen, worauf ich später noch eingehe (McCarthy 1960, S. 203):

> „A programming system called LISP (for LISt Processor) has been developed for the IBM 704 computer by the Artificial Intelligence Group at M. I. T. The system was designed to facilitate experiments with a proposed system called the Advice Taker, whereby a machine could be instructed to handle declarative as well as imperative sentences and could exhibit ‚common sense‘ in carrying out its instructions. [...] The main requirement was a program-

ming system for manipulating expressions representing formalized declarative and imperative sentences so that the Advice Taker system could make deductions."

Ähnlich wie im „Advice Taker"-Papier dieselbe Sprache für die Programmierung und die Kommunikation mit dem System, die Ein- und Ausgabe verwendet werden soll, wird LISP 1960 als etwas allgemeineres als eine Programmiersprache vorgestellt: als ein Formalismus, der für eine Theorie des Berechnens[140] gegenüber dem Turingmaschinenformalismus Vorteile haben, der aber auch für die Programmierung gut geeignet sein soll (McCarthy 1960, S. 203):

„In this article, we first describe a formalism for defining functions recursively. We believe this formalism has advantages both as a programming language and as a vehicle for developing a theory of computation."

LISP eignet sich McCarthys Darstellung zufolge zwar auch für die Programmierung, seine Gestalt scheint sich jedoch wesentlich aus theoretischen und von der Rechenmaschine unabhängigen Überlegungen zu ergeben, was ich im folgenden darstelle. Als Formalismus, mit dessen Hilfe Funktionen rekursiv definiert konnten, schien LISP mit der Rechenmaschine kaum noch etwas zu tun zu haben.

Heute wird LISP häufig als Umsetzung des λ-Kalküls in eine Programmiersprache verstanden. McCarthy selbst schreibt jedoch rückblickend, daß er die Überlegungen von Church zum Zeitpunkt der LISP–Entwicklung nicht verstanden hatte und lediglich dessen Notationsweise für Funktionen übernahm (McCarthy 1981, S. 176):

„To use functions as arguments, one needs a notation for functions, and it seemed natural to use the λ-notation of Church (1941). I didn't understand the rest of his book, so I wasn't tempted to try to implement his more general mechanism for defining functions."

Auf den Zusammenhang LISP – λ-Kalkül gehe ich deshalb nicht näher ein.

Die Funktionen, für die sich McCarthy in „Recursive Functions" interessiert, sind solche über „symbolic expressions", also symbolischen Ausdrücken (kurz „S-expressions").[141] Diese entsprechen im Grunde Listen, also einer Sequenz von elementaren Symbolen – in LISP-Terminologie Atomen – und Unterlisten, nur daß sie paarweise geordnet sind. Eine Liste der Form (a,b,c) z. B. entspricht dem S-Ausdruck (a·(b·(c·NIL))), wobei NIL das Ende der Liste anzeigt. Funktionen werden in dem Papier, da sie über „S-expressions" arbeiten, „S-functions" genannt. McCarthy

---

140 McCarthy verwendet in dem Papier durchgehend den Begriff „theory of computation" und nicht etwa „theory of computability"; es ging ihm also um eine Theorie des Berechnens. Im Gegensatz zu einer Theorie der Berechenbarkeit geht es in einer Theorie des Berechnens – folgt man der Begrifflichkeit selbst – nicht nur um die Frage danach, ob eine Zahl oder Funktion überhaupt berechnet werden kann, sondern auch darum, wie sie berechnet wird.

141 McCarthy entwickelt in diesem Artikel verschiedene Repräsentationsweisen von Funktionen, um zu ermöglichen, daß Funktionen auf Funktionen angewendet werden können. Die folgende Darstellung dieses Papiers mag auf Grund dieses Spiels mit den Repräsentationen etwas unübersichtlich wirken. Das spiegelt den besprochenen Text und die dort behandelte Thematik wider.

definiert S-Funktionen hier, wie er selbst anmerkt, weitgehend mit Hilfe konventioneller mathematischer Notationsweisen für Funktionen und nennt diese Weise, S-Funktionen auszudrücken, „meta-expression“ (kurz: „M-expression“).

Zunächst führt McCarthy in dem Papier einige elementare S-Funktionen ein, aus denen er mit Hilfe konditionaler Ausdrücke und rekursiver Definition eine größere Klasse von Funktionen aufbaut. Von dieser behauptete er beiläufig und ohne einen Beweis zu führen, daß sie alle berechenbaren Funktionen umfasse. Da die elementaren S-Funktionen auch in dem LISP-System die grundlegenden Funktionen darstellen, führe ich sie kurz auf, wobei ich zum besseren Verständnis auf eine vollständige und formale Definition verzichte. Es handelte sich um die Funktionen

- „car“, die angewendet auf einen S-Ausdruck dessen erstes Element zum Ergebnis hat;
- „cdr“, die angewendet auf einen S-Ausdruck dessen zweites Element (was der Liste entspricht, die verbleibt, wenn man ihr erstes Element streicht,) zum Ergebnis hat;
- „cons“, die angewendet auf zwei S-Ausdrücke diese zu einem zusammenfügt;
- „atom“, die angewendet auf ein elementares Symbol bzw. Atom „wahr“ und ansonsten „falsch“ zum Ergebnis hat;
- „eq“, die angewendet auf zwei Atome „wahr“ zum Ergebnis hat, falls diese gleich sind, im Falle der Ungleichheit „falsch“.

Für S-Funktionen entwickelt McCarthy eine Schreibweise, die erlaubt, sie selbst als Listen, als „S-expressions“, zu repräsentieren. Dazu gibt er Regeln an, mit deren Hilfe die M-Ausdrücke, mit denen er die S-Funktionen definiert hat, in S-Ausdrücke umgewandelt werden können. Die Repräsentation einer S-Funktion als S-Ausdruck sieht dabei ihrer ursprünglichen Beschreibung als M-Ausdruck sehr ähnlich. So wird der M-Ausdruck

cons [car[x]; cdr[x]]

übersetzt in den S-Ausdruck

CONS, (CAR, X), (CDR, X)

S-Funktionen werden damit in der gleichen Weise repräsentiert wie die Werte, auf denen sie operieren. Das ermöglicht, die Repräsentationen dieser Funktionen selbst als Eingabe für eine S-Funktion zu verwenden.

Um zu zeigen, daß sein Formalismus ein Beitrag zu einer Theorie des Berechnens darstellt, entwickelt McCarthy in seinem Formalismus ein Äquivalent zu einer universellen Turingmaschine[142], die Funktion „apply“. Diese beschreibt er mit Hilfe eines M-Ausdrucks. Angewendet auf den S-Ausdruck einer S-Funktionen und eine Liste von

---

142 Die universelle Turingmaschine erhält die Beschreibung einer speziellen Turingmaschine (vgl. Abschnitt 2.2.2) als Eingabe und simuliert dann das Verhalten der speziellen Turingmaschine.

Argumenten liefert diese dasselbe Ergebnis, wie die Funktion selbst angewendet auf diese Argumente (McCarthy 1960, S. 208):

> „There is an S-function apply with the property that if f is an S-expression for an S-function f' and args is a list of arguments of the form (arg1, ..., argn), where arg1, ... , argn are arbitrary S-expressions, then apply[f; args] and f'[arg1; ... ; argn] are defined for the same values of arg1, ..., argn, and are equal when defined."

„Apply" verhält sich damit – angewandt auf den zu einer S-Funktion gehörigen S-Ausdruck und Eingabewerte – wie eine universelle Turingmaschine, angewandt auf eine kodierte Beschreibung des Verhaltens einer anderen Turingmaschine und Eingabewerte. McCarthy hält fest (McCarthy 1960, S. 203): „[We] then describe the universal S-function *apply* which plays the theoretical role of a universal Turing machine and the practical role of an interpreter".

Neben der theoretischen Bedeutung einer universellen Turingmaschine hat „apply" auch die praktische Bedeutung eines Interpreters. Diese Idee, „apply" (das seinerseits im wesentlichen durch eine Unterfunktion „eval" realisiert wurde) in Assembler zu programmieren und als Interpreter für die S-Ausdrücke von listenverarbeitenden Funktionen zu verwenden, hatte Steve Russel, einer der Programmier in der „Artificial Intelligence Group", nachdem er von McCarthys Überlegungen gehört hatte – eine Anwendung, die McCarthy so nicht im Sinn gehabt hatte (McCarthy nach Stoyan 1980, S. 177):

> „Steve Russel sagte, ‚Gib mal her, warum sollte ich nicht dieses **EVAL** programmieren [...]?', doch ich antwortete ihm, ‚Ho, ho, du verwechselst Theorie und Praxis, denn dieses **EVAL** ist zum Lesen gedacht und nicht fürs Rechnen [...]' Doch er ging und tat es [...]."

Nachdem Russel und einer der Studenten, D. Edwards, bei der Programmierung noch einige Fehler in der Funktion beseitigt hatten, war der neue Interpreter lauffähig. S-Ausdrücke von S-Funktionen konnten nun auf der IBM 704 bearbeitet werden. Da zu dieser Zeit ein Compiler, der eigentlich für die LISP-Programmiersprache (in LISP) geschrieben werden sollte, noch nicht annähernd fertiggestellt war, mußten LISP-Programme nun streng als S-Ausdrücke notiert werden – eine Notationsweise, an die man sich schließlich gewöhnte und die blieb (McCarthy 1960, S. 212):

> „A variant of the function apply [...] has been translated into a program APPLY for the IBM 704. Since this routine can compute values of S-functions given their descriptions as S-expressions and their arguments, it serves as an interpreter for the LISP programming language which decribes computation processes in this way."

Ein LISP-Programm war damit ein symbolischer Ausdruck, ein Wert, der der implementierten S-Funktion „apply" eingegeben wurde. Bis dahin waren LISP-Programme, wie in dem Memo skizziert, aus Anweisungen und Ausdrücken zusammengesetzt gewesen. Um zu ermöglichen, imperative Sätze auch weiterhin zu verwenden, wurde eine besondere Funktion eingeführt, die erlaubte, Zuweisungen und GO TO „statements" in ALGOL- oder FORTRAN-ähnlicher Notation zu benutzen.

Doch zurück zu „Recursive Functions“: Kurz nachdem McCarthy hier die Funktion „apply“ definiert hat, leitet er aus seinen bis dahin rein formal-mathematischen Überlegungen das Programmiersystem LISP ab. Zunächst zeigt er, wie man S-Ausdrücke im Rechenmaschinenspeicher als Listen, die ähnlich wie in IPL realisiert sind, darstellen kann. Nachdem er zuvor die elementaren S-Funktionen als M- bzw. S-Ausdruck repräsentiert hat, gibt er nun deren, wie er es nennt, „computer representation“ an – er zeigt kurz, wie sie in Assembler programmiert werden können. Die Assembler-Programme erscheinen so als eine von vielen möglichen Repräsentationen und haben bei ihm den gleichen Stellenwert wie ein mathematisch-formaler Ausdruck. Anschließend diskutiert McCarthy, wie aus den elementaren S-Funktionen mit Hilfe konditionaler Ausdrücke und Rekursion gewonnene S-Funktionen (und damit seiner Behauptung zufolge alle berechenbaren Funktionen) kompiliert werden können. Da er auch den Interpreter aus einer der universellen Turingmaschine gleichenden Funktion hergeleitet hatte, hatte er ein Programmiersystem – Programmiersprache und deren „Prozessor“ – aus einem Formalismus abgeleitet, der wie Turingmaschinen nach ihm genau die Klasse der berechenbaren Funktionen darstellen konnte. Die Rechenmaschine spielt für die Konzipierung des Systems in dem Papier keine Rolle. Das Programmiersystem erscheint als ein mathematischer Formalismus.

Diesen Abschnitt abschließend noch einige Worte zu der weiteren Entwicklung von LISP: Im März 1960, einen Monat vor „Recursive Functions“, war das erste LISP 1 - Manual am MIT fertiggestellt. Nach dem Erscheinen der IBM 709 wurde für diese eine LISP-Version herausgegeben. Im August 1962, dem Erscheinungsdatum des Manuals zu LISP 1.5, war LISP, wieder unter der Leitung von McCarthy, wesentlich verbessert worden. Der Versuch einer radikaleren Änderung wurde Ende 1963 am MIT unternommen. LISP sollte näher an ALGOL angebunden werden, da die LISP-Notation zu unübersichtlich erschien. Dieses ehrgeizige LISP II - Projekt wurde später aus Kostengründen wieder aufgegeben, so daß LISP II nie größere Bedeutung erlangte.

LISP 1.5 hingegen fand nach 1962 weite Verbreitung, wurde zunächst auf verschiedenen Rechnern implementiert und nach 1965 von unterschiedlichen Personen und Gruppen weiterentwickelt, so daß eine ganze Reihe Dialekte (z. B. Mac Lisp, Interlisp, ELISP, PSL) mit verschiedenen Schwerpunktsetzungen entstand.[143] Nach FORTRAN ist LISP die älteste Programmiersprache, die auch heute noch eingesetzt wird.

---

143 Diese Entwicklung schildern recht detailliert Gabriel und Steele (1993).

4.6.2.4 Zusammenfassung: LISP als universelle Sprache

Wie Newell und Simon den „Logic Theorist“ gestaltete auch McCarthy den „Advice Taker“ entsprechend der Vorstellung, die er vom menschlichen Problemlösen und besonders Lernen hatte. Allerdings standen bei ihm Fragen nach der Kommunikation mit dem „Advice Taker“ und damit verbunden der Darstellung und Repräsentation des Problembereichs stärker im Vordergrund. Das Hauptproblem war für ihn zunächst, wie man dem Programm etwas mitteilen kann, um ihm Lernen zu ermöglichen, und wie dabei „interesting changes of behavior“ in einfacher Weise ausgedrückt werden können (vgl. McCarthy 1958, S. 404). Newell, Shaw und Simon hingegen fokussierten stärker auf innere Prozesse, die Art und Weise, wie Informationen verarbeitet werden. Während Newell, Shaw und Simon den LT und seine Realisierung ausführlich diskutierten, beschrieb McCarthy in seinem „Advice Taker“-Artikel vor allem die Sprache, in der er zum einen Programmieren und zum anderen die Entscheidungssituationen darstellen wollte. In bezug auf die Art und Weise, wie der „Advice Taker“ arbeiten sollte, umriß er nur relativ unausgearbeitete Ideen.

Während IPL als Programmiersprache, mit der der LT programmiert werden sollte, vor allem eine flexible Verarbeitung von im Speicher abgelegten Informationen zum Ziel hatte, ging es bei der LISP-Entwicklung sehr viel mehr um Notationsformen, um die Art und Weise der Darstellung. Denn die LISP-Sprache sollte in der Lage sein, unterschiedlichste Bereiche zu beschreiben: Neben der Realisierung von „gesundem Menschenverstand“ auf dem Computer sollte sie dazu dienen, mathematische Probleme wie die Differentiation komfortabel zu beschreiben; zudem sollte sie einen Formalismus für die Theorie des Berechnens bereitstellen. Es ging also um eine Sprache, die zur Programmierung menschlicher Denkvorgänge geeignet ist; sie sollte zudem in einem mathematischen Sinne einfach und elegant sein und – im Gegensatz zu FORTRAN – nicht nur der Durchführung von Berechnungen auf dem Computer dienen, sondern selbst einen Beitrag zur mathematischen Theoriebildung darstellen. Darüber hinaus sollte die LISP-Sprache als Ein- und Ausgabesprache für LISP-Programme und für die Repräsentation von Situationen und Anweisungen geeignet sein. So sollte der „Advice Taker“ in der gleichen Sprache programmiert werden wie der, in der ihm Mitteilungen gemacht und Anweisungen gegeben werden sollten – und zwar so, wie sich McCarthy das bei einem Menschen vorstellte: mit Hilfe einer Mischung aus imperativen und deklarativen Sätzen. LISP sollte also eine Art universelle Sprache werden, in der mathematische Theoriebildung, die Programmierung von Berechnungen und des gesunden Menschenverstandes sowie die weitgehend vollständige Beschreibung von Situationen und das Anweisen der programmierten Maschine möglich war.

Notationsweisen und in diesem Sinne Sprachdesign bildeten also einen Schwerpunkt der LISP-Entwicklung. Dabei schien die Rechenmaschine für die Gestalt der Sprache so gut wie keine Rolle zu spielen: In der ersten größeren Präsentation der Sprache standen formal-mathematische Überlegungen im Vordergrund – ja diese nahmen McCarthy das Sprachdesign buchstäblich aus der Hand. McCarthys Überlegungen zu LISP als Formalismus für eine Theorie des Berechnens mündeten, ohne daß er das beabsichtigt hatte, in die Programmierung eines Interpreters durch zwei Mitarbeiter des Projektes und führten zur Fixierung der Form der Sprache – diese bestand nun nur noch aus „S-expressions", also symbolischen Ausdrücken. Aus der „algebraischen Sprache zur Manipulation symbolischer Ausdrücke" des Memos von 1958, in der die Zuweisung die wichtigste Anweisung sein sollte, wurde eine rein deklarative Sprache. LISP erscheint so als ein Programmiersystem, das ganz – sowohl in bezug auf die Gestalt der Sprache als auch den sie verarbeitenden Interpreter – Kind der mathematischen Theoriebildung ist.

McCarthy ging es in „Recursive Functions" nicht zuletzt um eine Theorie des Berechnens.[144] Daß Berechnen für ihn mehr bedeutete, als nur eine Rechnung auszuführen, und eng mit seiner Sichtweise auf den gesundem Menschenverstand verknüpft war, zeigt der Vergleich zwischen seiner Vorstellung einer mathematischen Wissenschaft einerseits und gesundem Menschenverstand andererseits. Über eine mathematische Wissenschaft schrieb er 1962 (McCarthy 1962, S. 21):

> „In a mathematical science, it is possible to deduce from the basic assumptions, the important properties of the entities treated by the science."

Gesunden Menschenverstand faßte er 1958 wie folgt (McCarthy 1958, S. 403):

> „We shall therefore say that a program has common sense if it automatically deduces for itself a sufficiently wide class of immedeate consequences of anything it is told and what it already knows."

Eine mathematische Wissenschaft ebenso wie der gesunde Menschenverstand sind demnach wesentlich dadurch charakterisiert, daß ausgehend von einer bestimmten Menge von Fakten oder Annahmen Konsequenzen hergeleitet werden. Gesunder Menschenverstand und eine mathematische Wissenschaft liegen so gesehen nicht weit voneinander entfernt.

---

144 Die Theorie des Berechnens sollte ihn auch in den folgenden Jahren beschäftigen. 1961 hielt er auf der Western Joint Computer Conference einen Vortrag mit dem Titel „A Basis for a Mathematical Theory of Computation", 1962 auf der IFIP-Konferenz in München einen weiteren „Towards a Mathematical Science of Computation". In beiden umreißt er Grundzüge einer Wissenschaft des Berechnens, in der die Rechenmaschine im Gegensatz zu theoretisch-formalen Fragen nur eine untergeordnete Rolle spielt. Eulenhöfer (1998b, S. 143ff.) geht auf die hier präsentierte Sicht auf das Feld des Berechnens und ihre Auswirkungen auf die Disziplin Informatik näher ein.

Verständlich wird vor diesem Hintergrund, wie die beiden Hauptziele der LISP-Entwicklung – eleganter mathematischer Formalismus für eine Theorie des Berechnens zu sein, in dem Berechnungen wie die Differentiation, aber auch gesunder Menschenverstand programmiert werden können – zusammenhingen. Menschliches Denken war für McCarthy wesentlich logisches Schließen und damit berechenbar. Beides war von daher in der selben Sprache beschreibbar. Auch McCarthys Wunsch, in der LISP-Sprache dem „Advice Taker" Situationen beschreiben und Anweisungen geben zu können, so daß dieser ähnlich lernen kann wie ein Mensch, paßt in diesen Orientierungsrahmen. Bereits 1956 schrieben Shannon und McCarthy, daß das menschliche Gehirn als endlicher Automat betrachtet werden kann.[145] Alles was das menschliche Gehirn – gedacht als Automat – erkennen und beschreiben kann, läßt sich mit Hilfe einer formal-logischen Sprache beschreiben. Zumindest all das, was der Mensch in der Welt erkennen kann, ist so gesehen vollständig formal beschreibbar.[146] Da LISP-Programme Werte repräsentieren und besonders in der Anfangszeit – da speicherplatzaufwendig und langsam – sehr teuer waren, charakterisierte Alan Perlis diese Sicht scherzhaft wie folgt (nach dem Hacker's Dictionary, Internet): „LISP programmers know the value of everything and the cost of nothing".

LISP-Programme werden häufig als sich selbst aufbauende Entitäten wahrgenommen. Diese Sicht beschreibt Turkle mit Bezug auf Hofstädters „Gödel, Escher, Bach" für Hacker am MIT wie folgt (Turkle 1984, S. 281):

> „Die Schönheit dieser Sprache liegt für den Hacker darin, daß sie vollkommen rekursiv und selbstbezüglich ist. Höhere Ebenen werden aus ihren eigenen Elementen aufgebaut, wie Eschers endlos aufwärts führende Treppen und Bachs ‚endlos ansteigende Fuge', die ständig in höhere Tonarten wechselt, bis sie, auf scheinbar unerklärliche Weise, genau dorthin wieder zurückkehrt, von wo sie ausgegangen ist."

Diese Sicht auf LISP-Programme liegt zum einen darin begründet, daß Rekursion, bei der eine Funktion sich gewissermaßen aus sich selbst heraus aufbaut, die zentrale Kontrollstruktur in LISP darstellt.[147] Doch auch in anderer Hinsicht spielt der Gedanke der Selbstbezüglichkeit in LISP eine große Rolle: So sollte bereits 1958 (s. o. Memo) nach McCarthys Wunsch der Kompiler für LISP in LISP geschrieben werden; LISP-Programme können LISP-Programme verarbeiten und sollten – wie der „Advice Taker" – auch selbst programmieren können. Beliebt ist auch, die LISP-Sprache durch

---

145 Diese Auffassung formulierten McCarthy und Hayes 1969 besonders prägnant (S. 466): „The physical world exists and already contains some intelligent machines called people."

146 In einer BBC-Sendung am 39. 8. 1973 formulierte McCarthy entsprechend (McCarthy nach Weizenbaum 1977): „Der einzige Grund dafür, daß es uns bisher nicht gelungen ist, jeden Aspekt der realen Welt zu simulieren, ist der, daß wir noch nicht über ein genügend leistungsfähiges Rechenprogramm verfügen. Ich arbeite zur Zeit an diesem Problem."

147 Ein Beispiel für eine rekursive Definition ist: $n! = 1$, falls $n = 0$; $n! = n\,(n-1)!$, sonst. $n!$ wird also durch Bezug auf $(n-1)!$ und so gesehen durch sich selbst definiert.

die Angabe der LISP-Funktion „eval“ (die im wesentlichen dem oben besprochenen „apply“ entspricht) zu beschreiben (so z. B. in McCarthy 1981).

Für die sich am MIT entwickelnde und mit LISP programmierende Artificial-Intelligence-Community hält Turkle fest (Turkle 1984, S. 312): „Für die AI-Kultur, die sich am MIT um Minsky herum gebildet hat, besteht das reinste Vergnügen darin, Welten aus reinem Denken zu entwickeln.“ Da es Ziel der KI ist, Denken zu programmieren, ist „Denken konstruiert Denken“ nach ihr in dieser Kultur eine zentrale Aussage, der fast religiöse Bedeutung beigemessen wird. Bereits in „Recursive Functions“ spielte die Rechenmaschine für den Entwurf von LISP keine Rolle. Das findet seine Entsprechung darin, daß LISP-Programme später als sich aus sich selbst heraus aufbauend wahrgenommen wurden. Das Programmieren mit LISP erscheint als rein symbolische Denktätigkeit, der die Kraft zur Konstruktion anderen Denkens innewohnt.

4.7 Sprache und Programmiersprachen

Ich werde nun die besprochenen Programmiersprachenentwicklungen in Hinblick auf gemeinsame Orientierungsmuster, die mit der Verwendung des Begriffs Sprache zusammenhängen, auswerten. Für die Wahrnehmung des Programmierens – so die These, die ich dabei ausführe – verlor die Rechenmaschine mit der Übernahme der Sprachbegrifflichkeit mehr und mehr an Bedeutung. Ich beginne damit, telegrammartig die verschiedenen Weisen, in denen der Sprachbegriff in den besprochenen Programmiersprachenentwicklungen benutzt wurde, zusammenzustellen.

Vor ca. 1955 war der Gebrauch des Begriffes „Sprache“ in der Programmierung noch die Ausnahme und hatte metaphorischen Charakter. Bis 1960 war der Terminus Sprache zum Normalfall geworden. 1957 wurde die „FORTRAN language“ vorgestellt, die der „Sprache der Mathematik“ als sehr ähnlich angesehen wurde. Zudem war hier von „Maschinensprache“ die Rede; 1958 benutzte Backus auch den Begriff „hardware language“. Wenn auch vorsichtig wurden hier, wie später auch in der ALGOL-Entwicklung, Parallelen zwischen dem Übersetzen natürlicher Sprachen und dem von Programmiersprache in Maschinensprache hergestellt.

1956 hatten Newell und Simon eine „logische Sprache“ – 1957 dann „informationsverarbeitende Sprache“ genannt – beschrieben. Hier handelte es sich nach ihnen um „formale Sprachen“, die für die Spezifikation von Informationsverarbeitungssystemen wie auch deren Realisierung auf dem Computer geeignet waren. 1957 benutzten sie auch den Begriff „Programmiersprache“ für eine Menge von Symbolen und Konventionen, die erlauben, dem Computer zu spezifizieren, welche Prozesse ausgeführt werden sollen.

In dem ersten ALGOL-Report von 1958 war Sprache ein konstitutiver Begriff, der in verschiedensten Varianten vorkam: so z. B. als Computersprache, Programmiersprache, algebraische Sprache, Problemsprache oder Publikationssprache. Zudem war im Rahmen der ALGOL-Entwicklung von einer arithmetischen, algebraischen, algorithmischen und einer universellen Sprache die Rede. 1959 gab Backus eine formale Beschreibung der Syntax dieser Sprache und stellte eine baldige formale Behandlung der Semantik in Aussicht. In dem ALGOL-Report von 1960 wurde die Syntax und Semantik der Sprache unterschieden.

1958 hatte McCarthy in groben Zügen eine formale Sprache entworfen, die neben der Programmierung dazu dienen sollte, dem programmierten System, das nach ihm „verbal argumentieren" können sollte, Beschreibungen und Aufforderungen mitzuteilen. 1960 stellte er die LISP-Programmiersprache vor, die bei ihm als mathematischer Formalismus zur Beschreibung der berechenbaren Funktionen erschien. 1962 stellte er erste Überlegungen zur formalen Beschreibung der Semantik von Programmiersprachen an.

1959 schließlich wurde mit COBOL der Versuch unternommen, eine „Wirtschaftssprache" zu entwickeln, die der natürlichen Sprache möglichst ähnlich sein sollte. Die Rede war von „English language compilers", und man hoffte, sie neben der Programmierung auch zur Beseitigung von Verständigungs- und Kontrollproblemen zwischen Programmier- und anderen Abteilungen eines Unternehmens einsetzen zu können.

Es gab nun für die Programmierung algebraische, arithmetische oder algorithmische Sprachen, deren große Ähnlichkeit mit mathematischen Formalismen und Notationsweisen nicht in Frage gestellt wurde. Zudem gab es informationsverarbeitende Sprachen und quasi natürliche Sprachen. Programme, die in einer Sprache geschrieben werden und aus „statements", „declarations", „verbs", „sentences", oder „expressions" zusammengesetzt sind, wirken wie ein Text, der im Gegensatz zu der Instruktionenfolge von Neumanns nicht mehr so sehr anweisenden, sondern beschreibenden Charakter hat. Programmieren – das Aufschreiben dieses Textes – erscheint damit weniger als das Anweisen der Maschine denn als Darstellung oder Erläuterung. Die Maschine ist beim Aufschreiben dieses Textes weniger präsent.

Neben der Programmierung schienen die besprochenen Sprachen auch dazu geeignet zu sein, das jeweilige Problemfeld in einer für Menschen komfortablen Weise zu beschreiben, wurden also in einem recht buchstäblichen Sinne als Sprachen verstanden. Sie wurden auch als Publikationssprache gedacht (ALGOL), sollten zur Formulierung von psychologischen Theorien (IPL), zum Gespräch mit der Maschine (LISP) oder als Quasi-Englisch der Verständigung in Unternehmen (COBOL) dienen. Deutlich wird daran, daß maschinell-formale und menschlich-natürliche Sprache als eng verwandt angesehen wurden – was nicht zuletzt auch daran deutlich wird, daß die Übersetzung zwischen Programmier- und Maschinensprache der Übersetzung

zwischen natürlichen Sprachen als sehr ähnlich vorgestellt wurde. Der Entwicklung von Programmiersprachen unterliegt eine Sicht auf Sprache, wie ich sie in Abschnitt 4.1 dargestellt habe: Sprache erscheint als regelhaft aufgebaut und als auf Grundlage dieser Regeln künstlich herstellbar. Ähnlich wie bei Chomsky (1956) eine Grammatik durch ein Gerät oder organisch im Menschen realisiert sein kann, scheint Sprache in der Entwicklung der frühen Programmiersprachen von Mensch und Maschine gleichermaßen verstanden (und in der LISP-Entwicklung auch gesprochen) werden zu können. 1959 hielt Gorn auf der „International Conference on Information Processing" in Paris in der einleitenden Rede der Sektion „Common symbolic language for computers" fest (Gorn 1959, S. 117):

> „From the moment that von Neumann first suggested that the instructions and data should share the same storage in a machine there has been a growing realization that general purpose digital computers are linguistic mechanisms [...], and these mechanisms are now recognized to be capable of simulating the self-referencing subtleties of speech and thought."

Bei von Neumann wurde die Rechenmaschine durch die Folie des als Mechanismus gedachten Nervensystems verstanden. Nun erscheint sie vor dem Hintergrund einer als regelhaft herstellbar gedachten Sprache als linguistischer Mechanismus. Sie scheint die selbstreferentiellen Subtilitäten der Sprache – die Eigenschaft der natürlichen Sprache, die erlaubt, in ihr über sie selbst nachzudenken und zu sprechen – nachahmen zu können. Diese Einschätzung wurde besonders bei IPL und LISP deutlich, die erlauben sollten, daß die Maschine lernt und sich selbst verbessert. Die Rechenmaschine wird nicht mehr als Rechenmaschine, sondern gewissermaßen als Sprachmaschine – als Informationsverarbeitungssystem – verstanden.

Während von Neumann und Goldstine für das Programmieren eine erste Phase vorgesehen hatten, die mit der Maschine nichts zu tun haben sollte – die der mathematischen Problembeschreibung, der die eigentliche Kodierung erst folgte –, schien sich die Programmierung mit Sprachen auf diese Phase zu reduzieren. Die Spezifikation eines Problems und dessen Realisierung auf dem Computer wurden, wie das Newell und Simon ausdrückten, als im wesentlichen dieselbe Aufgabe gesehen. Die zweite Phase der Kodierung schien von der Maschine im wesentlichen selbst übernommen zu werden, wie das besonders in dem FORTRAN-Text sehr deutlich wird. Die Rechenmaschine schien mit der Programmierung in Sprachen ähnlich wenig zu tun zu haben wie mit der ersten Phase der mathematischen Problemvorbereitung bei Goldstine und von Neumann.

Da das Erstellen der mathematischen Problembeschreibung in der Regel die Aufgabe von Wissenschaftlern und Ingenieuren war, wurde mit der Einführung problemorientierter Sprachen wie FORTRAN, ALGOL, LISP und COBOL die Programmierung immer häufiger zur Aufgabe dieser Personengruppe. Zur Programmierung machten sich diese eine Sprache zu eigen, die auch zur Darstellung ihres Problembereichs – unter Umständen zur Formulierung von Theorien – geeignet schien,

aber doch durch die Erfordernisse der Rechenmaschine strukturiert war. Nötig wurde ein Blick auf das Problemfeld, den von Neumann (1948) in seiner Automatentheorie als „Analyse des Problems der Organismen und Organisation vom Standpunkt dieser relativ kleinen, künstlichen Automaten" charakterisiert hatte. Der Problembereich mußte in klar unterscheidbare Elemente und Unterelemente, zeitliche Prozesse mußten in diskrete, aufeinander folgende Abschnitte zerlegt werden. Da die Programmiersysteme wie selbstverständlich als Sprachen – oft auch mit Ähnlichkeit zur natürlichen Sprache – gesehen wurden, ist anzunehmen, daß die Erfordernisse der Programmierung einer digitalen Rechenmaschine weitgehend unbemerkt in verschiedene Wissenschaftsgebiete wie die Naturwissenschaften, die Mathematik, die Wirtschaftswissenschaften und die Psychologie eindrangen und dort die Problemwahrnehmung strukturierten.

Während der Code bei von Neumann auch als technisches Artefakt gedacht wurde und die einzelnen Instruktionen auch für jeden Durchlauf der Maschine durch sie standen, werden Programme mit der Einführung von Sprachen in die Programmierung mehr und mehr als sprachlich oder symbolisch gedacht; der technische, an die Maschine gebundene Charakter eines Programms tritt in den Hintergrund. Programmieren erscheint als symbolische Tätigkeit, das Programm vor allem als Text; die Eigenschaft von Programmen, technisches Artefakt zu sein, wird unsichtbarer. Von den technischen Artefakten bleibt jedoch, daß Programme als operierend gedacht werden. Sie erscheinen als operierende Notationen, deren Operationen – wie z. B. bei dem „FORTRAN translator" oder dem „logic theorist" – auch als Äquivalent zu menschlichen Tätigkeiten begriffen werden. Vor diesem Hintergrund erscheinen die Operationen als mehr oder weniger selbständige Handlungen und Programme nicht nur als operierende, sondern auch als handelnde Notationen. Programme erhalten ihre Bedeutung im Spannungsfeld zwischen Mensch, Maschine und Symbolsystem: Sie können als beschreibender Text, als maschinell operierende oder auch selbständig handelnde Notationen verstanden werden.

Kapitel 5: Schlüsse

Ich möchte nun die Ergebnisse der vorangegangenen Kapitel aus verschiedenen Perspektiven und mit unterschiedlicher Zielsetzung zusammenfassen. Zunächst (Abschnitt 5.1) betrachte ich die Herausbildung der Programmierung und der Programmiersprachen, wie ich sie in den vorangegangenen Kapiteln beschrieben habe, im Zusammenhang. Dabei arbeite ich die Art und Weise, wie der Code und Programme als hybride Objekte konzipiert wurden, und den damit verbundenen Orientierungsrahmen heraus. Besonders mache ich auf den Einfluß aufmerksam, den ein behavioristischer Handlungsbegriff in der Programmierung hat.

In Abschnitt 5.2 frage ich danach, welche Rolle die Hybridobjekt-Sichtweise in der heutigen Diskussion um die Softwareentwicklung spielt. Insbesondere zeigt sich dabei, daß ein behavioristischer Handlungsbegriff in methodischen Verfahren wie z. B. einem an Dijkstra angelehnten Modell der Softwareentwicklung und in dem JAVA-Objektbegriff reproduziert wird.

In dem letzten Abschnitt (5.3) weite ich mein methodisches Vorgehen von der Textinterpretation zu einem umfassenderen Ansatz aus, der einen konstruktiven Umgang mit gesellschaftlich-kulturellen Einflüssen auf wissenschaftlich-technische Geneseprozesse in der Informatik und Programmierung erlaubt. Insbesondere eignet sich der vorgestellte Methodenrahmen dazu, der Sinnhaftigkeit von Handeln in der Softwareentwicklung Rechnung zu tragen.

5.1 Hybridobjekte und die behavioristisch-atomistischen Sichtweise in der Programmierung

In den vorangegangenen Kapitel ist wiederholt deutlich geworden, daß im Diskurs um die Programmierung zunächst der Code bzw. später Programme in ein Sinngeflecht eingebettet wurden, das erlaubte, sie in zweierlei Weisen zu betrachten: Zum einen wurden sie als mathematisch-formale Beschreibung interpretiert; zum anderen wurde Programmen und den sie bildenden Einheiten in den besprochenen Texten auch die Kraft, zu operieren und selbständig zu handeln, zugesprochen. Sie wurden als agierende Notationen, als Hybridobjekte, konzipiert.

Daß Programme diesen doppelten Charakter als beschreibend und operierend zu haben scheinen, kann daraus erklärt werden, daß die Programmierung ihre Wurzeln zum einen in der Mathematik, zum anderen in der Elektrotechnik hat. Auf diesen doppelten Ursprung der Programmierung (und der Informatik) ist häufig hingewiesen worden. Die vorangegangenen Kapitel zeigen, daß für die Herausbildung der Programmierung auch Wissenschaftsdisziplinen von Bedeutung waren, deren

originäres Thema das menschliche Denken und Verhalten ist: die Neurophysiologie und der Behaviorismus, so wie er in den 30er und 40er Jahren in den USA in populärwissenschaftlicher Form weit verbreitet war. Das möchte ich in diesem Abschnitt, der die Ergebnisse der Kapitel 2, 3 und 4 in Zusammenhang bringt, aufzeigen. Zuvor fasse ich den Argumentationsgang kurz zusammen.

Neben allen Unterschieden macht die Interpretation der von mir behandelten Texte auch ein gemeinsames Element deutlich: Ihnen unterliegt eine Sichtweise, die ich die behavioristisch-atomistische Perspektive genannt habe. Diese Perspektive erlaubt, Prozesse und materielle Entitäten in diskrete Einheiten aufzusplitten, deren Verhalten eindeutig beschreibbar scheint. Sie ermöglicht, daß das Programmieren zu der universellen Technik wurde, als die es heute erscheint: Als programmierbar gelten zeitliche Prozesse – wie Arbeitsabläufe und menschliche Denkprozesse – aber auch Welten, materielle Entitäten und Organismen. Während einerseits die behavioristisch-atomistische Perspektive das Programmieren als eine mächtige Methode zur Konstruktion unterschiedlichster Aspekte der Welt auf dem Computer konstituiert, wurde mit der behavioristischen Sicht auch ein reduktionistischer Handlungsbegriff, der Handeln aus dem Ablauf von Reiz-Reaktions-Abläufen erklärt, in die Programmierung importiert.

Wie ich in Kapitel 2 dargelegt habe, entwarf von Neumann 1945 in dem „First Draft of a Report on the EDVAC" ein „high speed automatic digital computing system", das in seiner Darstellung mal als elektronische Maschine, mal als nervensystemartiger Organismus erscheint. Als konstitutive Elemente seines Geräts bzw. Entwurfes wählte er hypothetische Elemente, die erlaubten, von dem materiellen Aufbau zu abstrahieren. Sie konnten zwei Zustände besitzen, arbeiteten in einem bestimmten Zeittakt und wurden scheinbar sowohl der Sicht auf das Gerät als Maschine als auch der als Organismus gerecht: Sie wiesen nach von Neumann im wesentlichen das gleiche Ein-/Ausgabeverhalten sowohl menschlicher Neuronen wie auch von Vakuumröhren auf. Zusammengesetzt aus diesen hypothetischen Elementen erschien das „Rechensystem" des „First Draft" als operierende Struktur, die dem maschinellen wie auch dem menschlichen „Ausführen von Instruktionen zur Durchführung komplexer Berechnungen" unterlag.

Der Vergleich der Rechenmaschine mit dem menschlichen Nervensystem und Gehirn, wie ihn von Neumann hier und auch in der Folgezeit anstellte, war keine singuläre Erscheinung, sondern Ausdruck und Teil einer breiteren wissenschaftlichen Debatte der Zeit. So hatte Alan Turing 1936 zur Fassung der berechenbaren Zahlen hypothetische Maschinen vorgeschlagen, die nach ihm auch den menschlichen Rechenprozeß beschrieben. Er zerlegte eine Berechnung in eine Folge diskreter, elementarer Ziffernoperationen, die ihm zufolge von Mensch wie Maschine gleicher-

maßen ausgeführt werden konnten. Ähnlich wie von Neumann elementare Schritte des Rechnens in hypothetischen Elementen verortete, die für ein bestimmtes Ein-/Ausgabeverhalten standen, wurden bei Turing die elementaren Schritte von einer „Maschine“ ausgeführt, von der er ebenfalls nur das Ein-/Ausgabeverhalten betrachtete. Der materielle Aufbau und die Art und Weise, wie dieses Verhalten und die Elementaroperationen des Rechnens materiell realisiert werden sollten, war in Turings mehr noch als in von Neumanns Text kein Thema. Das Unterteilen des Rechenprozesses in elementare, voneinander abgrenzbare und regelhafte Akte erlaubte, das menschliche Rechnen als Vorgang zu beschreiben, der in der selben Weise auch von einer Maschine durchgeführt werden konnte.

Im Anschluß an Turing entwarfen McCulloch und Pitts, von denen von Neumann 1945 die Idee der E-Elemente übernahm, 1943 ein Modell des menschlichen Nervensystems und Gehirns, das dieses als machtvolles logisches Rechengerät, als Mechanismus, erscheinen ließ. Dazu modellierten sie die Funktionsweise von Neuronen in Modellen, die genau zwei Zustände einnehmen konnten. Auch für ihre Sicht auf das menschliche Gehirn war das Unterteilen von Denkprozessen in elementare, klar voneinander abgrenzbare Elementaroperationen, die ein wohldefinierbares Ein-/Ausgabeverhalten aufwiesen, konstitutiv. McCulloch und Pitts zeigten, wie die Unterteilung von Rechenprozessen in elementare Akte in ein Bild des menschlichen Nervensystems überführt werden kann, nach dem dieses aus Elementen besteht, die solche Elementarakte ausführen. Ein scheinbar in diskrete Einheiten unterteilbarer Denkprozeß wurde bei ihnen übersetzt in ein entsprechendes Bild von der diesen Prozeß ausführenden materiellen Entität. Da McCullochs und Pitts Elemente nicht den materiellen Aufbau, sondern das Ein-/Ausgabeverhalten von Neuronen modellierten, konnte von Neumann diese Elemente zur Beschreibung der speicherprogrammierbaren Rechenmaschine verwenden. Die elementaren Rechenoperationen von Turings „Maschinen“ waren so gewissermaßen über den Umweg neurophysiologischer Theorie zu hypothetischen Bausteinen des von Neumannschen Rechensystems geworden.

Ebenfalls 1943 veröffentlichten Rosenbluth, Wiener und Bigelow ein Papier, in dem sie explizit vorschlugen, was in obigen Papieren im Grunde umgesetzt wurde: eine behavioristische Sicht auf natürliche Ereignisse. Diese sollte darin bestehen, die Beziehung der Ausgabe eines sich verhaltenden Objektes zur Eingabe zu untersuchen. Maschinen wie lebende Organismen konnten nach ihnen gleichermaßen einer solchen behavioristischen Analyse unterzogen werden. Qualitative Differenzen zwischen maschinellem und dem Verhalten lebender Organismen waren nach ihnen nicht festzustellen.

Für den Januar 1945, einige Monate vor dem Erscheinen des „First Draft“, organisierten Wiener, von Neumann und Aiken ein Treffen von Wissenschaftlern aus unterschiedlichen Bereichen, zu dem auch McCulloch eingeladen war. Das Treffen mündete

in den Entschluß, nach dem Krieg eine neue Wissenschaft einzurichten, die ingenieurwissenschaftliche und neurophysiologische Aspekte vereinen sollte. Die behavioristische Betrachtung von natürlichen Ereignissen etablierte sich und sollte zur Grundlage einer neuen Wissenschaft werden, die später von Wiener Kybernetik genannt wurde. Die Art und Weise, wie von Neumann die speicherprogrammierbare Rechenmaschine im „First Draft" darstellte – als operierende, aus hypothetischen Elementen bestehende Struktur, die maschinellem und auch menschlichem Ausführen von Instruktionen zum Durchführen von Berechnungen unterlag – ist also Ausdruck eines sich in den USA in dieser Zeit etablierenden Denkstils. Charakteristisch ist für diesen eine behavioristische Sicht auf Organismen und Maschinen: Diese scheinen im wesentlichen das gleiche Verhalten – gefaßt als das Verhältnis der Ein- zur Ausgabe – aufzuweisen. Ein weiteres Kennzeichen ist ein Blick auf das Verhalten und Denken, nach dem dieses in elementare, klar definierbare Akte unterteilbar ist. Auch diese finden in behavioristischen Modellen ihre Entsprechung: in den Reflexen, die – in Muskeln und Nervensystem verankert – aus behavioristischer Sicht eine Handlung konstituieren.

Mit der Übernahme von McCulloch und Pitts Neuronenmodellen in den Entwurf und die Beschreibung einer Rechenmaschine verlegte von Neumann den behavioristischen Blick auf das Verhalten von Maschinen und Organismen als Ganzes – wie ihn Rosenbluth Wiener und Bigelow 1943 dargestellt hatten – auf eine tieferliegende Schicht: auf das Verhalten der Elemente, aus denen diese aufgebaut waren. In seiner Automatentheorie arbeitete er 1948 dieses Sicht auf Maschinen und Organismen weiter aus. Er faßte Organismen wie Maschinen als mehr (Organismen) oder weniger (Maschinen) große Anzahl sie konstituierender Elemente. Diese Elemente betrachtete er als „Automatismen", die auf einen bestimmten „Stimulus" mit einer bestimmten „Response" reagieren. Das Nervensystem erschien aus dieser Sicht wie das Rechengerät des „First Draft" als Struktur von in ihrem Verhalten vollständig und eindeutig beschreibbaren Elementen; aus einer mathematischen Forscherperspektive konnte es als Symbolsystem konzipiert werden. Die Organismen als Ganzes zu untersuchen erschien bei von Neumann auf Grundlage einer behavioristischen Sicht auf scheinbar klar isolierbare und regelhaft funktionierende Elemente entsprechend als Aufgabenbereich der Mathematik und Logik.

Auch die interne Organisation und damit wesentliche Aspekte des Aufbaus maschineller und besonders auch biologischer Körper können aus der behavioristisch-atomistischen Perspektive in der Terminologie von Reiz und Reaktion erklärt und als Symbolsystem gefaßt werden. Die Frage nach der Hervorbringung und dem „Wie" der materiellen Realisierung des Verhaltens der Organismen als ganzes kann so zunächst durch das Zusammenwirken elementarer Elemente erklärt werden, von denen lediglich ihr scheinbar vollständig und eindeutig beschreibbares Ein-/Ausgabeverhalten

betrachtet wird. Von außen beobachtbares Verhalten wird so gesehen durch elementarere Verhaltensschritte realisiert, die als hypothetische Ein-/Ausgabeelemente oder mathematische Funktion gefaßt werden können. Dieses Vorgehen kann man auch weiter fortsetzen: Stellt man sich die Elemente der ersten Verfeinerungsstufe ihrerseits als zusammengesetzt aus weiteren, ebenfalls als Stimulus-Response-Automatismen beschreibbaren Elementen vor, kann auch ihr Verhalten in dieser Weise beschrieben werden usw. Die jeweils elementareren Verhaltensschritte bringen so gesehen das jeweils übergeordnete Verhalten hervor. Die Frage nach der materiellen Realisierung des Verhaltens kann durch den behavioristischen Blick auf atomare Elemente zurückgedrängt und weitgehend durch die Beschreibung unterliegender, als hypothetische Elemente faßbarer Verhaltensformen beantwortet werden. Die Frage nach der materiellen Umsetzung verwandelt sich weitgehend in die Frage nach dem Zusammenwirken von vollständig beschreibbaren Verhaltensschritten. In von Neumanns Automatenbegriff spiegelt sich wider, daß aus einer behavioristischen Perspektive die Differenz zwischen technisch und natürlich verschwimmt, und daß aus einer behavioristischen Perspektive auf atomare Bestandteile die Grenze zwischen körperlichem Organismus und Symbolsystem – und in diesem Sinne zwischen materiell und immateriell – undeutlich wird. Automaten konnten Maschinen, Organismen aber auch mathematische Konstrukte – Symbolsysteme – sein, wie das schon bei dem Gerät des „First Draft" der Fall war.

Die behavioristisch-atomistische Perspektive steht im Zentrum des Orientierungsrahmens, vor dessen Hintergrund die Programmierung elektronischer Rechenmaschinen entstand und beschrieben wurde. Begreift man den behavioristisch-atomistischen Blick als wichtigen Bestandteil der Programmierung, ist entsprechend zum einen – wie ich in Abschnitt 3.2.2 herausgearbeitet habe – eine Charakterisierung der Tätigkeit des Programmierens möglich: Der behavioristisch-atomistische Blick erlaubt, den zu programmierenden Gegenstandsbereich als Automaten – als Struktur aus Ein-/Ausgabeelementen – zu konzipieren. Im Prozeß des Programmierens kann diese Struktur qua Verfeinerung in einen technischen Automaten – die programmierte Maschine – überführt werden. Zum anderen wird vor dem Hintergrund der behavioristisch-atomistische Perspektive verständlich, warum in der Programmierung mehr als nur eine Tätigkeit zur Steuerung der Rechenmaschine gesehen werden kann und sowohl das menschliche Denken als auch die Entstehung des Lebens als programmiert betrachtet werden können.

Die behavioristisch-atomistische Perspektive konstituierte eine Wahrnehmung des Programmierens, die bis heute für die Assoziationen, Handlungsentwürfe und technisch-wissenschaftlichen Visionen – die Orientierungsschemata –, die mit der Programmierung zusammenhängen, von Bedeutung ist. Das heißt nicht, daß das

Programmieren zwangsläufig mit einem solchen Orientierungsrahmen verbunden sein muß. Genauso wenig ist jedoch möglich, es von heute auf morgen von den entsprechenden Assoziationen zu befreien, zumal sie zumindest in den von mir betrachteten ersten zwanzig Jahren in der Programmierung immer wieder eine wichtige Rolle spielten. Nachdem ich in den vorangegangenen Ausführungen den Einfluß der behavioristisch-atomistischen Sichtweise auf die Wahrnehmung der Rechenmaschine zusammengefaßt habe, werde diesen Orientierungsrahmen nun in Hinblick auf die Programmierung genauer beschreiben.

Wie in den vorangegangenen Kapiteln deutlich wurde, ermöglichte ein behavioristischer Handlungsbegriff in Zusammenhang mit einer behavioristisch-atomistischen Sichtweise, Organismen wie Maschinen und insbesondere das menschliche Nervensystem bzw. Gehirn und die Rechenmaschine in bezug auf ihr äußeres Verhalten wie auch ihre interne Organisation als im wesentlichen gleichartige Gebilde zu konzipieren. Die Instruktionen, aus denen der Code bestand, richteten sich so gesehen an ein Gerät, das als ein Gegenstück zum menschlichen Nervensystem und Gehirn erschien. Das „Ausführen von Instruktionen, um komplexe Berechnungen durchzuführen" – das von Neumann 1945 als die Funktion seines Rechensystems beschrieb – ist so gesehen, egal ob von der Maschine oder dem Menschen durchgeführt, im Grunde die gleiche Tätigkeit. Damit scheint zumindest prinzipiell gleichgültig zu sein, wer instruiert bzw. programmiert wird – ein menschlicher oder ein maschineller Rechner. Der Unterschied zwischen einer mathematischen Beschreibung, die für einen Menschen bestimmt ist, und dem die Maschine steuernden Artefakt erscheint marginal. Auch wenn in den von Neumannschen Texten zur Programmierung thematisiert wird, daß eine mathematische Problembeschreibung nicht einfach in den Code übersetzt und der Code nur in engem Zusammenhang mit der Maschinenfunktion verstanden werden kann (vgl. Abschnitt 2.4.2), legt die Wahrnehmung der Rechenmaschine als eine Art Nervensystem nahe, Code und mathematische Problembeschreibung als prinzipiell gleichartig zu sehen. Eine solche Wahrnehmung des Codes spielte in der Folgezeit eine entscheidende Rolle: In der Entwicklung von Programmiersprachen wurden Unterschiede zwischen einem Programm und einem für Menschen bestimmten Text nicht mehr thematisiert.

In dem Vergleich Rechenmaschine-Gehirn bei von Neumann drückt sich ein Orientierungsmuster aus, das den Diskurs der Programmierung entscheidend beeinflußte: Die Rechenmaschine wurde als eine Art Elektronengehirn gedacht, das so oder zumindest ähnlich wie ein Mensch rechnet. Wie besonders der McCulloch/Pitts-Text zeigt, schien sich Denken im wesentlichen im Rechnen zu erschöpfen; die Rechenmaschine schien damit nicht nur wie ein Mensch zu rechnen, sondern wenigstens prinzipiell auch wie ein Mensch zu denken. Der Versuch, auch andere menschliche Denkprozesse als das Rechnen zu programmieren, lag also nahe. Umgekehrt wurde möglich, das Programmieren auch als Technik zur Steuerung des

menschlichen Denkens zu begreifen, wie sich das bis heute in populären Bemerkungen wie „der ist wohl falsch programmiert", „ich muß mich mal anders programmieren", aber auch in therapeutischen Verfahren wie NLP (Neurolinguistik Programming) oder zumindest deren Benennung ausdrückt.

Die behavioristisch-atomistische Sicht auf Organismen lag auch den von von Neumann 1948 angestellten Überlegungen zur Selbstreproduktion von Automaten zugrunde, die von Neumann auch mit der genetischen Reproduktion in Verbindung brachte (vgl. Abschnitt 3.1.2). Die Kodier- und Programmierterminologie wurde in der Folgezeit in die biologischen Wissenschaften importiert: Leben erschien als in Genen kodiert bzw. programmiert, die zum einen als vollständige Beschreibung, zum anderen als die Entstehung des Lebens steuernde Kraft gedacht wurden, ähnlich wie von Neumann Instruktionen zum einen beschreibende, aber auch bewegende und steuernde Eigenschaften zuschrieb. Für die Programmierung bedeutete das im Umkehrschluß, daß sie nun auch als eine Tätigkeit in der Natur zur Steuerung der Entstehung lebender Organismen erschien. Die Begriffe Programm und Programmieren waren und sind bis heute mit Assoziationen umgeben, die sie mit natürlichen Vorgängen in Verbindung bringen. Das Programmieren der Rechenmaschine ist so auch konnotiert als Nachbildung eines natürlichen Vorgangs, der lebende Organismen hervorbringt. Während Programme einerseits zu beschreiben scheinen oder auch als Theorie angesehen werden, werden sie andererseits auch als Objekte, Welten oder auch Übersetzer oder Theorembeweiser – als quasi-materielle und auch handelnde Entitäten – gedacht. Wie aus einer behavioristisch-atomistischen Sicht Organismen technisch erklärbar werden und als Symbolsystem konzipiert werden können, können umgekehrt Programme – Notationen – als eine Art handelnde Organismen gedacht werden. Ähnlich wie in der Genetik wird auch im Bereich der Programmierung das Verhältnis Repräsentant – Referenz neu gedacht (vgl. Abschnitt 3.2.1). Das Programm scheint nicht nur zu beschreiben – zu repräsentieren –, es scheint bis zu einem gewissen Grad das Beschriebene – die Referenz – auch zu sein, wie das am Beispiel des „logic theorist" deutlich wird: Während das Programm einerseits als theoretisch-formale Beschreibung eines Problemlösungsprozesses gesehen wurde, stellt der „logic theorist" gleichzeitig einen Problemlöser dar, der scheinbar wie ein Mensch mathematische Sätze beweist.

In den von mir besprochenen Texten zur Entwicklung von Programmiersprachen, die Mitte der 50er Jahre ihren Anfang nahm, zeigt sich, daß hier das Programmieren als universellere Tätigkeit als das Steuern einer Rechenmaschine gesehen wurde. Vor der Einführung von Programmiersprachen wurde das zu programmierende Problem zunächst mathematisch-formal aufbereitet und anschließend von professionellen Kodierern programmiert. Programmiersprachen sollten nun dazu dienen, daß die mit einem entsprechenden Programm ausgerüstete Rechenmaschine die Kodierung

weitgehend selbst übernahm. Die frühen Programmiersprachen erschienen entsprechend als den Sprachen weitgehend gleich, in denen eine mathematische Problembeschreibung bzw. eine Spezifikation erstellt werden kann: Programmiersprachen wurden als mathematische Sprachen (FORTRAN) für die Publikation numerischer Algorithmen (ALGOL) oder die innermathematische Theoriebildung (LISP), als formale Sprachen zur Formulierung von Theorien über menschliche Denkprozesse (IPL) oder – im Bereich der wirtschaftlichen Datenverarbeitung – als quasi natürliche Sprachen gesehen (COBOL). Auf Grundlage eines Sprachverständnisses, nach dem Sprachen auf Grundlage von grammatikalischen Regeln auch künstlich hergestellt werden können, schien die Rechenmaschine annähernd die gleichen Sprachen verstehen zu können, wie der Mensch – die Programmiersprachen konnten entsprechend an den Menschen wie an die Maschine gerichtet sein und schienen in beider Hinsicht nützlich.

Vollzogen in einer Sprache erschien das Programmieren weniger als Steuern durch Befehle, denn als Beschreiben oder Feststellen durch Sätze oder Ausdrücke. Programme, die in einer Sprache geschrieben waren, die der mathematischen oder auch der natürlichen Sprache als ähnlich galt, wurden als mathematische Formel oder formale Theorien begriffen. Das Programmieren erscheint als rein mathematisch-formale oder sprachliche Tätigkeit. Vergessen wird dadurch nur allzu leicht, daß Programmieren eine Tätigkeit zur Steuerung einer Maschine ist, und daß die verwendeten Sprachen und Ausdrucksmittel diesem Zweck genügen müssen. Das birgt die Gefahr in sich, die Struktur der Rechenmaschine auf die Welt zu projizieren und den zu programmierenden Gegenstandsbereich wie einen Automaten funktionierend wahrzunehmen, ohne sich des Modellcharakters dieser Vorstellung und der Konsequenzen dieser Perspektive bewußt zu sein.

Programme bzw. ihre Einheiten wurden in den von mir zur Entstehung von Programmiersprachen analysierten Texten einerseits als beschreibend gesehen; andererseits erschienen sie auch, wie zuvor von Neumanns Instruktionen, als agierende Entitäten, wurden also als Hybridobjekte gedacht. Von heute gesehen wirft diese doppelte Sicht auf Programme als mathematische Beschreibung und operierender Gegenstand die Frage auf, woher eine Beschreibung oder auch ein Gegenstand die initiative Kraft zu einer absichtsvollen Handlung nimmt, wie sie das Steuern darstellt. Diese Frage verweist auf die Grenze der Erklärungsmacht des behavioristisch-atomistischen Vorgehens und verdeutlicht eine Problematik, die im Diskurs der Programmierung bis heute virulent ist: Mit dem Import des behavioristischen Blicks in die Programmierung war einerseits möglich, das Programmieren als eine fast universale Technik zur Konstruktion verschiedenster Aspekte der Welt im Computer zu konzipieren. Dem Programmieren unterliegt jedoch ein behavioristischer Handlungsbegriff, nach dem Handeln aus dem Ablauf von Reiz-Reaktionsmechanismen

erklärbar ist. Die Frage nach dem Sinn des Handelns oder einer Operation und damit verbunden nach den Motiven, aus denen die Handlung resultiert, bleibt damit außen vor; in der behavioristischen Theorie ist sie nicht beantwortbar.

Scheinbar selbsttätig agierende Programme verbergen entsprechend die Menschen, die den Sinn einer Handlung oder Operation herstellen; zum einen diejenigen, die Programme in einer bestimmten Absicht und mit einem bestimmten Ziel niedergeschrieben haben. Selbst handelnde Programme erscheinen nicht so sehr als Produkt einer Person oder Gruppe, sondern als selbständige Einheiten. Zum zweiten lassen als agierend gedachte Programme die Rolle der Menschen, die mit den Ergebnissen dieser Tätigkeit konfrontiert werden, leicht in Vergessenheit geraten. Zwar sind sie es, die diese Ergebnisse mit Sinn und Bedeutung ausstatten müssen; ein als handelnd gedachtes Programm scheint jedoch, wie ein handelnder Mensch, selbst Bedeutung in dieses Ergebnis zu legen. In kritischen – und in der Regel nicht als Teil der Disziplin geltenden – Betrachtungen der Informatik werden entsprechend besonders zwei Themen immer wieder angesprochen, die die Menschen sichtbar machen, die hinter als selbsttätig gedachten Programmen allzuleicht verschwinden: einerseits die Verantwortung der Informatiker, die hinter den Programmen stehen; andererseits die Einbettung der programmierten Systeme in den sozialen Kontext und damit die Rolle der Menschen, die die programmierten Systeme nutzen und ihre Operationen und Ergebnisse interpretieren und in einen sinnvollen Zusammenhang stellen müssen. In einem Diskurs der Programmierung, der von Beginn an eng mit behavioristischen Sichtweisen und einem behavioristischen Handlungsbegriff verbunden war, sind solche Themen, die nach Sinn und Bedeutung der programmierten Operationen fragen, nur schwer ansprechbar.

5.2 Computer und Programmierung heute

In diesem Abschnitt geht es mir darum aufzuzeigen, daß die Orientierungsmuster, die der Programmierung in ihrer frühen Zeit unterlagen, bis heute reproduziert werden. Auf Grundlage der in 5.1 dargestellten Überlegungen zur Rolle einer behavioristisch-atomistischen Sichtweise in der Entstehung der Programmierung frage ich nach den Spuren dieser Sichtweise im heutigen Diskurs der Programmierung. Dazu betrachte ich ein Modell der Softwareentwicklung, das besonders akzentuiert Dijkstra vertreten hat, sowie anschließend den JAVA-Objektbegriff. Dabei zeigt sich, daß diesen Konzepten implizit ein behavioristischer Handlungsbegriff unterliegt. Das hat Auswirkungen auf die Personen, die selbst programmieren, und auch diejenigen, die mit den programmierten Systemen umgehen müssen. Zudem führt es dazu, daß Programme – hier Objekte – als handelnde Wesen gedacht werden können.

Daß eine behavioristisch-atomistische Sichtweise und damit zusammenhängend ein behavioristischer Handlungsbegriff im Diskurs der Programmierung bis heute eine Rolle spielt, drückt sich z. B. in der Dijkstraschen Sicht auf die Softwareentwicklung aus. Nach dieser bildet eine formale Spezifikation für ein Programm den Ausgangspunkt der Programmierung und kann auch einem mathematisch geführten Korrektheitsbeweis dienen. Das Programmieren wird hier in einem Bereich angesiedelt, der rein formaler Natur zu sein scheint. Vor der Programmierung wird eine formale Spezifikation erstellt; mit Hilfe formaler Methoden wird aus dieser ein Programm erstellt. Der Erfolg der Programmierung – die Korrektheit des Programms – wird schließlich an der formalen Spezifikation gemessen.

Doch ist es wirklich möglich, das Programmieren so in einem formalisierbaren Bereich einzuschließen, ohne daß dabei auch Annahmen über die tatsächliche Welt einfließen und in diese zurückwirken? Der Versuch, diesen Einfluß genauer aufzuschlüsseln, führt zunächst zu den Programmierenden. Peter Naur hat die Rolle der Programmierenden in einem so konzipierten Softwareentwicklungsprozeß 1985 wie folgt beschrieben (S. 47):

> „[...] much current discussion of programming seems to assume that programming is similiar to industrial production, the programmer being regarded as a component of that production, a component that has to be controlled by rules of procedure and which can be replaced easily. [...] Such views agree well with the notion, seemingly common among persons working with computers, that the human mind works like a computer."

Die Programmierenden werden also als „Komponenten" im Produktionsprozeß betrachtet, die wie Stimulus-Response Mechanismen nach definierten Regeln arbeiten. Ihre Arbeit ist demnach auf Grundlage eines behavioristischen Handlungsbegriffs – zumindest theoretisch aber mit Auswirkungen auf den Arbeitsalltag – erklär- und planbar. Naurs Bemerkung, daß viele mit Computern arbeitende Personen davon ausgingen, daß das menschliche Gehirn wie ein Computer funktioniere, deutet darüber hinaus an, daß die Sicht auf das Nervensystem als Automat im Bereich der Programmierung auch über die Gestaltung der Programmierarbeit hinaus von Bedeutung ist.

Auch betroffen von einem solchen Softwareentwicklungskonzept sind die Nutzenden des Systems. Ähnlich wie der Arbeitsbereich der Programmierung als regelgeleitet konzipiert wird, ist ein solches Konzept der Softwareentwicklung in einem Einsatzbereich sinnvoll, der auf Grundlage explizierbarer Regeln beschrieben werden kann. In diesem Fall kann davon ausgegangen werden, daß die benötigte Funktionalität formal beschrieben und aus dem Kontext herausgelöst in ein Programm umgesetzt werden kann, das anschließend in Form einer programmierten Maschine in den ursprünglichen Kontext zurückgegeben wird. In einer „Automatenwelt" – wie sie bei technisch eingebetteten Systemen gegeben ist – und auch vor dem Hintergrund eines behavioristischen Handlungsbegriffs erscheint das nicht als Problem; im

wesentlichen wird ein Reiz-Reaktions-Automatismus durch einen anderen ersetzt. Geht man jedoch von einem anderen Handlungsbegriff aus, nach dem Handeln nicht losgelöst von seinem Sinn verstanden werden kann, erscheint dieses Vorgehen besonders für den Computereinsatz in menschlichen Arbeitsumgebungen problematisch. Das zeigt z. B. die Kritik von Christiane Floyd an einem solchen Verständnis der Programmierung – von ihr „Produktionssicht" genannt. Sie hält fest (Floyd 1993, S. 12f.):

> „Die Produktionssicht reduziert jedoch das Vorfeld der Anforderungsermittlung auf Vorgaben für zu modellierende Objekte und Systemfunktionen. Sie beschränkt Mensch-Rechner-Interaktion auf Systembedienung und kann die Einbettung der Software in menschliches Arbeitshandeln nicht thematisieren. [...] Da die Produktionssicht aber die Einordnung von Software in menschliche Sinnzusammenhänge systematisch ausblendet, verfremdet sie die Softwareentwicklung."

Was Floyd von Anhängern der Produktionssicht unterscheidet, ist vor allem ihr Handlungsbegriff. Für Floyd ist der Sinn einer Handlung und damit die Frage nach dem spezifischen Verständnis einer Systemfunktion durch die Nutzenden in ihrem Arbeitskontext ein zentraler Faktor in der Softwaregestaltung und -nutzung. Dijkstra, ein prominenter Vertreter der Produktionssicht, hingegen geht davon aus, daß die Nutzung eines Systems allein auf Grundlage der Kenntnis seiner Eigenschaften erfolgen kann, die nach ihm formal spezifiziert werden können und damit ohne Bezug zum Einsatzkontext existieren (Dijkstra 1977 zitiert nach Naur 1989, S. 474):

> „[...] a computer program is a tool that one can use by virtue of the knowledge of its explicitly stated properties. Those stated properties are known as ‚its functional specification' or ‚its specification' for short."

In diesem Bild der Nutzung hat die Frage nach dem Sinn und Kontext der Nutzung keinen Platz. Es erscheint deshalb unproblematisch, daß die Programmierung in einem eigenen, aus dem Nutzungskontext losgelösten Bereich stattfindet. Floyd hingegen konzipiert den Softwareentwicklungsprozeß als kooperativen Erkenntnisprozeß zwischen Entwicklern und Nutzern, an dem spätere Nutzer teilhaben sollen. Deutlich wird daran, daß Dijkstras Konzept der Programmierung innerhalb eines eigenen, formal behandelbaren Rahmens gerade dadurch ein bestimmtes Nutzerbild mit Auswirkungen auf reale Nutzer produziert, weil die Programmierung in einem eigenen, aus dem späteren Einsatzkontext herausgelösten Bereich stattfindet. Nutzende werden mit einem fertigen System konfrontiert, dessen Sinn in ihrem Arbeitsalltag sie anschließend weitgehend selbst herstellen müssen.

Daß ein behavioristischer Handlungsbegriff in einem an Dijkstra angelehnten Konzept eine Rolle spielt, läßt auch der hier verwendete Begriff von Korrektheit eines Programms vermuten. Der Korrektheitsbeweis erhält in einer behavioristisch-atomistischen Sicht auf den zu programmierenden Bereich größere Bedeutung als nur der Abgleich zwischen zwei formalen Definitionen – dem Programm und der funktionalen Spezifikation – zu sein. In einer Automatenwelt, in der die Schnittstellen zwischen

zwei Systemen und auch die auf die Ausgabe eines Systems erfolgenden Reaktionen klar definierbar sind, beschreibt die funktionale Spezifikation weitgehend das tatsächliche Funktionieren des programmierten Systems in seinem Einsatzgebiet. Ein Korrektheitsbeweis zeigt – vorausgesetzt ein behavioristischer Handlungsbegriff, der erlaubt, den menschlichen Benutzer als Automaten zu modellieren –, daß das programmierte System in seinem Einsatzfeld im wesentlichen korrekt arbeitet. Der hohe Aufwand eines Korrektheitsbeweises erhält vor dem Hintergrund einer behavioristischen Sichtweise auf den Einsatzkontext so eine zusätzliche Rechtfertigung. Rückt man jedoch die Frage nach dem Sinn in das Zentrum der Aufmerksamkeit, wie das z. B. Dirk Siefkes (1993) mit der Frage „Wann ergibt ein Beweis Sinn?" (S. 98) getan hat, so scheint ein solcher Beweis zu dem eigentlichen Problem, der Frage nach dem korrekten Funktionieren des Programms in seinem Einsatzkontext, nur wenig beizutragen (Siefkes 1993, S. 106):

> „Es gibt Informatiker, die ein Programm ohne Korrektheitsbeweis für wertlos halten. Da Programme formal sind, gebrauchen sie formale Programmiermethoden und machen so aus dem Programmieren eine mathematische Tätigkeit. Das mag nützlich sein, um abstrakte Programmiersprachen zu erklären. Aber das Problem der Kommunikation wird dadurch nicht gemildert. [...] Richard DeMillo, Richard Lipton und Alan Perlis haben schon 1979 in ihrer Arbeit ‚Social Processes and Proofs of Theorems and Programs' gemeint, formale Beweise seien keine große Hilfe, um andere von der Korrektheit unserer Programme (und Theoreme) zu überzeugen."

Die bisherigen Ausführungen zusammengefaßt ergibt sich, daß – obwohl bei einer an Dijkstra angelehnten Sicht auf die Softwareentwicklung das Programmieren scheinbar in einem formalisierbaren Bereich eingeschlossen wird – Annahmen über die reale Welt, in der die Programme erstellt und eingesetzt werden, einfließen. Um die Programmierung vom Einsatzkontext loslösen zu können, wird eine Struktur des Einsatzgebiets angenommen, die dieses Herauslösen möglich macht: Die Funktionalität des zu programmierenden Systems muß im Vorhinein mathematisch exakt beschreibbar sein. Zudem muß das fertige System nach seiner Fertigstellung in diesen Bereich integrierbar sein. In einem mit Hilfe der Automatenterminologie beschreibbaren Einsatzkontext ist dieses Vorgehen problemlos. Auch wenn menschliche Arbeitsprozesse betroffen sind, ergeben sich dann zunächst keine Schwierigkeiten, wenn diese auf Grundlage eines behavioristischen Handlungsbegriffs erklärt werden können oder man zumindest glaubt, sie könnten so erklärt werden. Probleme werden dann sichtbar, wenn man das behavioristische Bild des Handelns in Frage stellt. Dann wird deutlich, daß die Dimension des Sinns, die eine bestimmte Arbeitshandlung oder auch Funktionalität eines Computerprogramms in seinem Einsatzkontext hat – und damit weitgehend nicht explizierbare Grundlagen des Handelns –, in diesem Modell ausgeblendet wird.

Daß ein behavioristischer Handlungsbegriff auch heute in der Programmierung eine Rolle spielt, wird subtiler an der aktuellen Rede von tätigen Programmen, Objekten usw. deutlich. Notationen werden hier wie selbstverständlich als agierend gedacht, wobei Agieren das automatische Erfolgen einer bestimmten Reaktion auf eine bestimmte Eingabe meint. In der Konzeption von selbstagierenden Notationen, von Programmen als hybriden Objekten, die sowohl zu beschreiben als auch zu agieren scheinen, wird also ein behavioristischen Bild des Handelns reproduziert. Stellt man den behavioristischen Handlungsbegriff in Frage, wird deutlich, daß die Rede von selbstagierenden Notationen und Programmen eine habituell verankerte Handlungspraxis darstellt. Handeln wird hier wie selbstverständlich auf den Ablauf von Reiz und Reaktion reduziert. Daß Programmen bis heute die Fähigkeit etwas zu tun zugeschrieben wird, trägt also die Spuren behavioristischer Weltanschauung in sich – und auch die Gefahr, daß ein behavioristischer Handlungsbegriff ebenso selbstverständlich auf andere Bereiche als den der Programme übertragen wird. Um das zu verdeutlichen, werde ich nun den Objektbegriff in JAVA ausführlicher diskutieren.

Java wurde ab 1990 ursprünglich als Programmiersprache für die Software von Elektrogeräten – Toastern, Mikrowellenherden, Kaffeemaschinen – bei Sun entwickelt. Da die Hersteller die Prozessoren für solche Geräte häufig wechseln, muß die entsprechende Software auf den neuen Prozessoren laufen, sobald diese eingeführt werden – benötigt wurde damit eine architekturunabhängige Sprache. Als 1993 das World Wide Web im Internet erschien und sich sehr schnell durchsetzte, erkannten die Java-Entwickler, daß ihre Sprache für die Programmierung im Internet ideal geeignet wäre, weil ein Java-Programm auf allen im Netz angeschlossenen Rechnern würde laufen können. Als erste Anwendung wurde deshalb ein Webbrowser (HotJava) geschrieben. Kleine Java Programme – Java Applets – konnten nun dazu verwendet werden, eine Webseite mit interaktivem, ausführbarem Inhalt zu versehen, z. B. mit einer rotierenden Kaffeetasse. In der Folgezeit und bis heute war und ist Java in der fachöffentlichen Diskussion präsent. Mittlerweile wird es an vielen Universitäten im Informatik-Grundstudium gelehrt.

Im Oktober 1995 veröffentlichten James Gosling, der führende Entwickler von Java, und Henry McGilton im World Wide Web auf den Sun Seiten „The Java Language Environment. A White Paper". Dieses Papier wird in fast allen Büchern zu Java zitiert und war eines der ersten, das Java einer breiten Öffentlichkeit vorstellte. Beschrieben werden hier die grundlegenden Konzepte von Java, wobei C-Kenntnisse auf Seiten der Lesenden vorausgesetzt werden – der Text richtet sich also an relativ versierte Programmiererinnen und Programmierer. Weil er breit rezipiert wurde, aus der Entwicklungsgruppe selbst heraus für ein technisches Publikum entstand, und zudem zu den ersten Java-Veröffentlichungen zählt, bildet er die Grundlage der folgenden Ausführungen. Dabei konzentriere ich mich auf den Abschnitt, in dem die

grundlegenden Konstrukte der Programmiersprache Java, die Objekte, motiviert und beschrieben werden. Dort heißt es (Gosling, McGilton 1995, S. 33f.):

> „3.2 What are objects?
>
> At its simplest, object technology is a collection of analysis, design, and programming methodologies that focuses design on modelling the characteristics and behavior of objects in the real world. True, this definition appears to be somewhat circular, so let's try to break out into clear air.
>
> What are objects? They are software programming *models*. In your everyday life you're surrounded by objects: cars, coffee machines, ducks, trees, and so on. Software applications contain objects: buttons on user interfaces, spreadsheets and spreadsheet cells, property lists, menus, and so on. These objects have *state* and *behavior*. You can represent all these things with software constructs called objects, which also can be defined by their *state* and *behavior*."

Der Begriff Objekt wird hier in verschiedenen Hinsichten verwendet: Zunächst werden Objekte als Modelle für die Programmierung eingeführt. Zweitens werden technische Konstrukte, Autos und Kaffeemaschinen, als Objekte bezeichnet. Drittens erscheinen auch Lebewesen wie Enten und Bäume als Objekte. Viertens sind Objekte Elemente von graphischen Benutzungsoberflächen auf dem Bildschirm. Und fünftens gelten auch Software-Konstrukte, d.h. Bestandteile des Programmtextes, als Objekte. Der Begriff „Objekt" wird also in fünf verschiedenen Bedeutungen verwendet: als Modell, als technisches Ding, als Tier oder Pflanze, als Element von graphischen Benutzungsschnittstellen und als Software-Konstrukt.

Da die verschiedenen Objekte alle Zustand und Verhalten haben, können sie den Autoren zufolge in gleicher Weise als Software-Konstrukte repräsentiert werden. Wie in von Neumanns Automatentheorie ermöglicht der Blick auf ein scheinbar klar definierbares Ein-/Ausgabeverhalten, biologische und technische Entitäten in derselben Weise zu modellieren, und schließlich die Verhaltensbeschreibungen und die realen materialen Gegenstücke unter einem Begriff, dem des Objekts, zu fassen. Ähnlich wie Automaten können Objekte technisch, biologisch oder Notation sein.

Daß hierbei ein behavioristischer Begriff des Handelns eine zentrale Rolle spielt, wird an folgendem Beispiel deutlich. Im Anschluß an obige Textstelle heißt es:

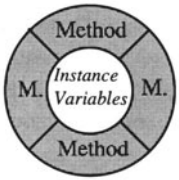
> »In your everyday transportation needs, a car can be modelled by an object. A car has state (how fast it's going, in which direction, its fuel consumption, and so on) and behavior (starts, stops, turns, slides, and runs into trees).«

In diesem Beispiel ist es das Auto, das startet und gegen Bäume fährt, nicht der Fahrer oder die Fahrerin. Das Auto scheint absichtsvoll zu handeln, der Mensch verschwindet hinter dem scheinbar selbst handelnden Artefakt. Verhalten, als das Ausführen einer vorbestimmten Operation, und Handeln, das mit Absichten verbunden ist, werden identifiziert, wie das für den behavioristischen Handlungsbegriff charakteristisch ist. Die Objekte scheinen autonom und absichtsvoll zu handeln, obwohl sie sich nur verhalten. Die Kategorie des Sinns oder der Absicht gerät so aus dem Blickfeld. Obwohl hier nur Enten als Beispiele aus dem biologischen Bereich angeführt sind, ist

damit die Gefahr groß, daß menschliches Handeln und maschinelles Verhalten bei der Programmierung objektorientierter Systeme gleich behandelt werden.

Ein zweiter Punkt wird anhand des Autobeispieles deutlich: Die Rede von handelnden Maschinen und Artefakten ist nicht auf den Bereich der Programmierung und der Informatik beschränkt. Ein behavioristischer Handlungsbegriff spielt in der Rede über Maschinen schon lange eine Rolle. Auch die Übertragung dieses Handlungsbegriffes auf die mit den Maschinen umgehenden Menschen und Menschen allgemein ist in diesen Bereichen eine Gefahr. Im Diskurs der Programmierung wurde die behavioristische Sichtweise auch auf das Denken und geistige Tätigkeiten übertragen. Die Besonderheit, daß mit dem Aufkommen von Computern auch geistige Tätigkeiten als mechanisch-maschinell verstanden wurden, ist bereits ausführlich und auch kritisch diskutiert worden. Ich habe in dieser Arbeit noch auf eine weitere Eigenheit hingewiesen, die ich unter dem Stichwort „atomistische Sicht" gefaßt habe: Im Zuge der aufkommenden Programmierung, deren Artefakte – Programme – unsichtbar auf der Maschine operieren, wurde ein behavioristischer Handlungsbegriff auch auf andere unsichtbare Einheiten wie Neuronen und Gene angewendet, deren Verhalten nun das Verhalten der sichtbaren Einheiten hervorzubringen schien. Die Entstehung des Lebens und das Programmieren wurde so in engen Zusammenhang gebracht.

Auch in dem JAVA-Papier findet diese Anwendung des behavioristischen Handlungsbegriffs auf unsichtbare Einheiten und die Assoziation „Programmieren – Entwicklung des Lebens" ihren Ausdruck. Objekte werden hier mit Zellen verglichen. Folgendes Diagramm dient den Autoren dazu, ein Objekt zu illustrieren:



Sie schreiben: „The diagram illustrates the conceptual structure of a software object – it's kind of like a cell, with an outer membrane that's its interface to the world, and an inner nucleus that's protected by the outer membrane" (S. 34). Software-Objekte werden also mit Zellen, d. h. den kleinsten, eigenständig lebensfähigen Einheiten aller Lebewesen, verglichen. Weitere aus dem Bereich der Biologie entlehnte Begriffe zur Darstellung von Objekten und objektorientierter Programmierung sind

- „Polymorphismus" (S. 33). In der Biologie meint dies das regelmäßige Vorkommen unterschiedlich gestalteter Lebewesen innerhalb derselben Art; die Java-Autoren benutzen ihn, um den Umstand zu benennen, daß dieselbe Nachricht, gesendet an unterschiedliche Objekte, zu unterschiedlichem Verhalten führen kann;

- „Vererbung". Bezeichnet wird hier die Eigenschaft von Klassen, Methoden an Objekte untergeordneter Klassen weitergeben zu können. Da nur Lebewesen vererben können, wird hier so etwas wie Lebendigkeit von Objekten nahegelegt – zumal Objekte auch eine „lifetime" haben und sich in gewissem Sinne auch fortzupflanzen scheinen: Objekte können neue Objekte kreieren, die u. U. eine längere Lebenszeit als sie selbst haben (vgl. S. 32, 34)

Wie es für zumindest höhere Lebewesen üblich ist, haben Java-Objekte scheinbar auch soziale Eigenschaften aufzuweisen: sie besitzen eine Art Privatsphäre, die „instance variables" („instance variables are private to the object"), die allerdings mit „befreundeten Klassen" geteilt oder „öffentlich" gemacht werden kann (S. 34). Objekte senden und empfangen Nachrichten, und können sie auch an Objekte senden, deren „Typ" sie nicht kennen (S. 33). Schließlich können sie sich frei bewegen: „objects can come from anywhere, possibly across the network" (S. 33).

Objekte erscheinen also als agierende Notationen und können auch für technische und biologische Entitäten stehen. Die Hybridobjekt-Sichtweise ist also in der Programmierung nach wie vor aktuell. Während die in den vorangegangenen Kapiteln angesprochenen Programme durch die Folie der Arbeit einzelner Menschen – wie z. B. das Rechnen, Übersetzen oder Beweisen – verstanden wurden, scheint es sich bei Objekten um eine Art elementarer Lebewesen zu handeln, die so etwas wie soziales Verhalten und Bewegungsfähigkeit aufweisen. Vor dem inneren Auge entsteht das Bild einer Gesellschaft aus Objekten, die kommunizieren, sich fortpflanzen und reisen. Nahe liegt damit, Objekte nicht nur – wie z. B. den „logic theorist" – als Modell und Theorie des Denkens des einzelnen Menschen zu sehen, sondern sie auch als Modell für die ganze Gesellschaft aufzufassen. Ein behavioristischer Handlungsbegriff wird damit auch in das Nachdenken über menschliche Gesellschaften und soziales Verhalten hineintransportiert.

Bevor ich im nächsten Abschnitt den methodischen Teil meiner Arbeit abrunde, möchte ich hier einige abschließende Reflektionen zum inhaltlichen Teil anstellen: Wie am Beispiel JAVA deutlich wird, sind hybride Objekte, agierende Notationen, bis heute im Diskurs der Programmierung von Bedeutung und vielleicht aktueller als in der Zeit zuvor. Der Computer und seine Programmierung, so wie wir sie bis heute denken, stellen damit althergebrachte Dichotomien in Frage: zunächst die zwischen biologisch und technisch und schließlich die zwischen Organismus und Symbolsystem und damit zwischen materiell und immateriell. Die Auflösung dieser Dichotomien erfolgt von zwei Seiten und endet in der Konzeption von hybriden Objekten, die Notation und Aktion in sich vereinen: Einerseits werden Programme und damit Texte als handelnde Gegenstände gedacht; andererseits wird der „Computer" als abstrakte Maschine vorgestellt, deren materieller Aufbau nur eine untergeordnete Rolle spielt.

Die Rede ist von der immateriellen, der logischen oder der Geistmaschine (Berr 1990, S. 198):

> „Mit dieser neuen Technik hat sich die technische Reproduktion von der Logik der determinierenden Materie befreit. Die neue Maschine wird als immaterielles System definiert, in dem die mechanischen Anteile völlig aus ihrer Funktionslogik verbannt werden."

Oder (Haraway 1995, S. 37 f.):

> „Die Maschinen des späten 20. Jahrhunderts haben die Differenz zwischen natürlich und künstlich, Körper und Geist, selbstgelenkter und außengesteuerter Entwicklung [...] höchst zweideutig werden lassen. Unsere Maschinen erscheinen auf verwirrende Weise quicklebendig - wir selbst dagegen beängstigend träge." .

Der Computertechnik wurde und wird oft eine Schlüsselstellung bei der Beantwortung der Frage nach der gesellschaftlichen Zukunft zugesprochen.[148] Begründet wird diese Wichtigkeit zumeist mit der technischen Funktionalität und einer sich daraus ergebenden Steigerung der Produktivität im Falle des Einsatzes von Computern. Ich denke, daß die Einschätzung des Computers als Schlüsseltechnologie auch aus den Sinnzusammenhängen und kulturellen Deutungen erklärt werden kann, die mit diesem Artfakt verknüpft sind: In den in dieser Arbeit besprochenen und für die Entwicklung der Rechenmaschine und ihrer Programmierung zentralen Texten werden die Grenzen zwischen technisch und natürlich sowie materiell und immateriell – zwischen technischem Gegenstand, Organismus und Symbolsystem – undeutlich, sie werden fallen gelassen und neu gedacht – ein Umstand, den ich mit dem Begriff Hybridobjekt-Sichtweise gefaßt habe. Der Computer und Programme gewinnen einen Teil ihrer Bedeutung so vielleicht nicht zuletzt in einer Beispielfunktion: Sie bieten ein Exempel, anhand dessen am Verlauf dieser Grenzen gerüttelt wird. In der Diskussion über Computer und im Begriff des Programms werden Eckpfeiler des abendländischen Denkens implizit mitverhandelt – und nicht zuletzt dadurch gewinnt diese Technologie ihre kulturelle Bedeutung und ihre für viele faszinative Kraft.

Ich habe in dieser Arbeit auch auf den Orientierungsrahmen hingewiesen, vor dessen Hintergrund sich die Hybridobjekt-Sichtweise konstituierte. Deutlich wurde, daß in den besprochenen Texten das Verschwimmen der Grenze zwischen technisch und natürlich sowie zwischen materiell und immateriell eine behavioristisch-atomistische Perspektive und einen behavioristischen Handlungsbegriff voraussetzte. Die gleichzeitig abstrakt-theoretischen und doch so anfaßbar scheinenden Objekte der Informatik sollten deshalb auch mit Vorsicht behandelt werden: Zwar scheinen mit ihnen alte Grenzen neu gedacht zu werden, allzuleicht reproduzieren und manifestieren sie jedoch ein mittlerweile überholtes Weltbild, in dem Handeln auf den Ablauf von Reiz und Reaktion reduziert wird.

---

148 Anhand der Diskussion um die Einführung der Informatik als Schulfach wird das anschaulich von (Kettnaker 1998) aufgezeigt, anhand der Debatte vor der Einführung der Informatik als Hochschulstudiengang in der Bundesrepublik Deutschland von (Eulenhöfer 1998b, S. 177ff.).

5.3 Die Explikation von Orientierungsmustern in der aktuellen Wissenschafts- und Technikentwicklung

Zum Abschluß möchte ich einige Überlegungen dazu anstellen, wie die Explikation von impliziten Wissensbeständen und Orientierungsmustern auch als Teil gegenwärtiger Theorie- und Technikentwicklung nutzbar gemacht werden kann. Da der Sinn einer Handlung sich weitgehend vor dem Hintergrund impliziten Wissens konstituiert, tragen solche Methoden einem Handlungsbegriff Rechnung, nach dem Handeln nicht losgelöst von seinem Sinn verstanden werden kann. Für die Programmierung habe ich aufgezeigt, daß vielen in diesem Bereich erstellten Konzepten ein behavioristischer Handlungsbegriff unterliegt. Die folgenden Ausführungen verstehen sich als Beitrag dazu, der Sinnhaftigkeit von Handeln in der Softwareentwicklung besser Rechnung tragen zu können. Ich integriere hier die Methode der dokumentarischen Interpretation technischer Texte, die ich im ersten Kapitel vorgestellt habe, in ein breiteres Methodenrepertoire. Obwohl ich mich vorwiegend auf die Softwareentwicklung beziehe, kann das vorgestellte Methodenrepertoire auch auf andere Bereiche von Wissenschaft und Technik übertragen werden.

Implizites Wissen stellt ein wesentliches Element in der wissenschaftlich-technischen Arbeit dar, wie ich im ersten Kapitel ausgeführt habe. Die Explikation impliziter Wissensbestände kann auf der einen Seite im Rahmen der wissenschaftlich-technischen Forschung, zum anderen auch in der Softwareentwicklung, wie sie im wirtschaftlichen Kontext betrieben wird, nutzbringend sein. Ich gehe auf diese beiden Bereiche, die natürlich miteinander zusammenhängen, ein.

Die Explikation impliziten Wissens im Rahmen der Wissenschaftsdisziplin Informatik kann zum Fortschreiten der Disziplin beitragen, neue, implizit schon vorhandene Denkräume öffnen und so weiterführende Erkenntnis ermöglichen. Denn wissenschaftliche Entdeckungen beruhen „immer auch auf der Explikation von impliziten Wissensbeständen und Denkstrukturen, wie sie in der kollektiven Geschichte der Disziplin über Generationen hinweg als implizite tradiert worden sind.“ (Bohnsack 1999). Eine Möglichkeit, im Rahmen der wissenschaftlichen Disziplin implizite Wissensbestände nutzbar zu machen, ist die Rekonstruktion von Orientierungsmustern, die vergangenen Theorie- und Technikentwicklungen unterlagen, ähnlich wie ich das in den vorangegangenen Kapiteln getan habe. Denn in der Vergangenheit hergestellte Sinnzusammenhänge wirken unter Umständen bis heute in die Theorie- und Technikentwicklung ein und werden im Rahmen neuerer Entwicklungen, die ja an ältere anschließen, immer wieder reproduziert. Am Beispiel der Hybridobjekte ist das deutlich geworden: Das Objektkonzept in Java z. B. reproduziert ein Sinngeflecht, das in der Informatik eine lange Tradition hat. Insbesondere unterliegt ihm ein

behavioristischer Handlungsbegriff, wie er allgemein im Rahmen der Programmierung eine wichtige Rolle gespielt hat und bis heute spielt. Ein solcher Handlungsbegriff legt – auch und gerade, wenn er implizit ist – für die Softwareproduktion ein bestimmtes Methodenrepertoire nahe. Die Explizierung des Handlungsbegriffs kann deshalb dazu beitragen, für bestimmte Softwareentwicklungskontexte bewußt von einem anderen Handlungsbegriff auszugehen, so zu neuen Methoden der Softwareentwicklung zu kommen und damit die disziplinäre Entwicklung voranzutreiben. Eine Gegenüberstellung Dijkstra-Floyd mag diesen Gedankengang verdeutlichen:

Im letzten Abschnitt habe ich aufgezeigt, daß die verschiedenen Sichten auf die Softwareentwicklung, die Dijkstra und Floyd vertreten, mit einem unterschiedlichen Handlungsbegriff einhergehen. Während im Dijkstraschen Modell ein behavioristischer Handlungsbegriff zentral ist, hebt Floyd auf die Sinnzusammenhänge ab, in denen Software entwickelt und eingesetzt wird. Der Handlungsbegriff, den ich mir in dieser Arbeit zu eigen gemacht habe, ist dem von Floyd sehr ähnlich. Sie geht davon aus, daß die Funktionalität eines Systems weder von den Entwicklern noch von den späteren Nutzern im Vorhinein beschrieben werden kann, sondern daß sich die Bedeutung einer Systemfunktion erst im Umgang mit dieser erschließt. Erst im Umgang mit dem System kommen demnach implizite Wissensbestände zum Tragen und wird der Sinn einer Systemfunktion deutlich. Ihr Modell der partizipativen Softwareentwicklung trägt dem Rechnung: Nutzer werden immer wieder mit Prototypen des zu erstellenden Systems konfrontiert; im Umgang mit diesen entwickeln sie ihre Anforderungen und formulieren ihre Wünsche aus.

Ich bin, wie ich im ersten Kapitel ausgeführt habe, davon ausgegangen, daß Handeln maßgeblich durch eine implizite Ebene des Wissens strukturiert wird. Explizit formulierte Handlungsentwürfe werden in dieser Sicht erst vor dem Hintergrund eines Orientierungsrahmens verständlich, der Ausdruck der Erfahrungen einer Person ist und in weiten Teilen nicht expliziert werden kann. Erklärt werden kann so, warum es für Softwareentwickler und Nutzer häufig so schwer ist, sich zu verständigen. Über die unterschiedlichen Fachsprachen hinaus sind diese in der Regel verschieden beruflich sozialisiert und in einen unterschiedlichen Arbeitsalltag eingebunden. Ihre Orientierungsrahmen sind von daher sehr verschieden. So kann es sein, daß Anwender und Entwickler sich über die Organisationsstruktur eines zu entwickelnden Systems scheinbar verständigen, die Ergebnisse dieser Verständigung jedoch vor dem Hintergrund ihrer jeweiligen Orientierungsrahmen unterschiedlich interpretieren und sich dadurch auf grundsätzliche Weise nicht verstehen. Das entstehende Produkt mag dann der Spezifikationen genügen und trotzdem seinen Sinn und Zweck nicht erfüllen.

Ich möchte im folgenden den Handlungsbegriff, auf den ich in dieser Arbeit rekurriere, auf den Bereich der Softwareentwicklung übertragen und aufzeigen, wie Methoden zur Explikation impliziter Wissensbestände zur Bewältigung des Verstän-

digungsproblems zwischen Nutzern und Anwendern beitragen können. Das Ziel dieser Methoden ist ein kontrolliertes Fremdverstehen, wie es z. B. auch in Teilen der sozialwissenschaftlichen Forschung ein wichtiges Anliegen ist. Zentrales Moment dieser Methoden ist, zu ermöglichen, daß die Untersuchten ihr eigenes Relevanzsystem und ihre eigene Perspektivität entfalten können. Das erlaubt, Zugang zu den Sinnstrukturen und dem Orientierungsrahmen zu finden, der ihrem Handeln unterliegt.

Für eine den Gestaltungsprozeß begleitende Analyse von Orientierungsmustern bieten sich folgende Methoden[149] an:

- *Narrative Experteninterviews*: Im Gegensatz zu z. B. Leitfadeninterviews werden narrative Interviews nicht durch Fragen vorstrukturiert. Im Vordergrund steht ebenfalls nicht das Abfragen von Argumentationen durch Fragen wie: „Warum arbeiten Sie in diesem Projekt?" oder „Weshalb finden sie diese Entscheidung schlecht?". Statt dessen werden erzählanregende Fragen gestellt, die auf Erfahrungen und Handlungen abzielen. Mögliche Eingangsfragen wären etwa: „Wie sind Sie dazu gekommen, mit Computern zu arbeiten?" oder „Wie erleben Sie ihre alltägliche Arbeit hier?". Ergänzt wird die Eingangsfrage durch erzählanregende Nachfragen, in denen der oder die Interviewte aufgefordert wird, bestimmte Aspekte der Eingangserzählung zu elaborieren. Solche Fragen fordern zu längeren Darstellungen auf, in denen die Befragten ihr eigenes Relevanzsystem (das durch einen Fragenkatalog vorgegeben würde) entwickeln können. Deutlich wird der Orientierungsrahmen, vor dessen Hintergrund die Aussagen der interviewten Person zu verstehen sind.

Mögliche Interviewpartner sind sowohl Auftraggeber, Anwender, aber auch Entwickler. Die Eingangsfrage muß auf die Rolle der jeweils interviewten Person im Entwicklungsprozeß abgestimmt werden. Die ggf. nur teilweise transkribierten Interviews werden auf unterliegende Orientierungsmuster hin analysiert. Diese können anschließend daraufhin geprüft werden, in welcher Weise sie in der Systementwicklung und für die Gestalt der entstehenden Technik relevant sind, und wie die expliziten Wünsche von Auftraggebern und Anwendern vor diesem Hintergrund zu verstehen sind. Dabei können auch Anforderungen an das zu erstellende System sichtbar werden, die noch nicht ausgesprochen wurden. Zudem können Problematiken deutlich werden, deren Lösung man sich durch den Computereinsatz erhofft, die aber nur auf andere Weise gelöst werden können. Bei der Einbeziehung auch von Entwicklern als Interviewpartner können Unstimmigkeiten zwischen den Orientierungen der Entwickler und Nutzer geprüft werden.

- *Gruppendiskussionen*: Gruppendiskussionen dienen dazu, die Orientierungsstruktur einer Gruppe und kollektive Orientierungsmuster zu erarbeiten. Ausgangspunkt für die Durchführung von Gruppendiskussionen ist zum einen die Erkenntnis,

---

149 Genauere Beschreibungen der angeführten methodischen Verfahren finden sich in (Bohnsack 1993).

daß latente Meinungen häufig erst im Gespräch und der Auseinandersetzung mit anderen Kontur gewinnen. Zum anderen wird mit dieser Interviewform dem Umstand Rechnung getragen, daß Meinungen und Haltungen häufig interaktionsabhängig, d. h. von der Gruppe, in der sich ein Individuum befindet, nicht zu trennen sind. Gruppendiskussionen können z. B. im Rahmen der Gruppe der Nutzer, der Entwicklungsgruppe oder auch in gemischten Gruppen durchgeführt werden, um Übereinstimmung und Unstimmigkeiten sichtbar zu machen.

Die einleitende Frage wie auch weitere Interventionen der Diskussionsleitung zielen hier darauf ab, eine eigendynamische und selbstläufige Diskussion zu erreichen. Die Fragen sollen dazu führen, daß gemeinsame Erfahrungen aktualisiert werden. Ein Beispiel für eine Eingangsfrage wäre: „Sie arbeiten doch hier an der Erstellung von System XY. Wie sind denn da so Ihre Erfahrungen?".

- *Teilnehmende Beobachtung*: Im Rahmen teilnehmender Beobachtung soll die Erarbeitung von Orientierungsmustern mithilfe obiger Methodiken ergänzt und mit der alltäglichen Praxis rückgekoppelt werden. Bei der teilnehmenden Beobachtung geht es darum, an dem zu untersuchenden Prozeß teilzunehmen, ohne sich zu engagieren; der beobachtete Gegenstand bzw. Ablauf soll möglichst wenig verändert werden. Die eigene, eventuell trotzdem manipulativ wirkende Rolle als beobachtend ist bei der Auswertung der Beobachtungen mitzureflektieren. Beispiele für den im Zusammenhang mit der Softwareentwicklung sinnvollen Einsatz dieser Methode sind die Beobachtung des Arbeitsalltages, bevor ein System eingeführt wird, die Beobachtung der Nutzung eines halbfertigen oder fertigen Systems oder die Teilnahme an Projektsitzungen, die während eines Entwicklungsprozesses stattfinden.

- *Textanalysen*: Mithilfe der dokumentarischen Interpretation können zum einen die transkribierten Interviews oder Gruppendiskussionen analysiert werden. Ergänzend kann sie dazu dienen, verschiedene, im Rahmen einer Systementwicklung erstellte Texte – z. B. Planungspapiere, Ergebnisse von Systemanalysen, Spezifikationen, Handbücher – auf unterliegende Orientierungsmuster zu untersuchen, wie ich das in Kapitel 1 methodisch beschrieben und in den folgenden Kapiteln anhand informatischer Fachtexte vorgeführt habe. Dabei kann erarbeitet werden, wie sich in Interviews deutlich gewordene Orientierungen in schriftlicher und öffentlicher Form wiederfinden und sich verändern. Besonders kann eine solche Analyse dazu beitragen, zu überprüfen, ob die für die Systemerstellung verbindlichen Papiere den Anforderungen und Orientierungen gerecht werden, die zuvor im Gespräch mit Auftraggebern und Anwendern herausgearbeitet wurden.

Ich bin in dieser Arbeit von einer Sichtweise auf die Wissenschafts- und Technikentwicklung ausgegangen, für die zentral ist, Handeln auch als Ausdruck von Sinnstrukturen und impliziten Wissensbeständen zu verstehen. Mithilfe eines metho-

dischen Vorgehens, das diesem Handlungsbegriff entspricht, habe ich für den Bereich der Programmierung herausgearbeitet, daß hier ein behavioristischer Handlungsbegriff eine zentrale Rolle spielt. Auch dieser Handlungsbegriff zieht entsprechende methodische Verfahren nach sich, die bis heute in der Programmierung von großer Bedeutung sind. Ich habe abschließend meinen Handlungsbegriff auf die Softwareentwicklung übertragen und Ansätze zu einem entsprechenden methodischen Vorgehen in diesem Bereich vorgestellt. Das verdeutlicht, was der Untertitel dieser Arbeit „Zur kulturellen Konstruktion der Programmierung" meint. Technik- und Wissenschaftsentwicklung baut auf grundlegenden Sichtweisen – z. B. die Sicht auf das Handeln – auf. Eine Änderung dieser Sichtweisen zieht ein anderes methodisches Vorgehen und damit andere Formen der Erkenntnis nach sich. In diesem Sinne ist Wissenschafts- und Technikentwicklung kulturell: Sie ist von Orientierungen und Deutungen durchdrungen, die auf der einen Seite ihre Einsichten ermöglichen, auf der anderen Seite auf die Schranken dieser Einsichten verweisen. Die Frage nach der Angemessenheit eines methodischen Vorgehens in einem bestimmten Bereich spielt von daher eine große Rolle: Welche Aspekte kann ich so erfassen, welchen werde ich weniger oder gar nicht gerecht, und was folgt daraus für die Anwendbarkeit einer Theorie bzw. den Einsatz einer Technik? Das mag sich zunächst wie eine moralisierende Beschränkung der wissenschaftlich-technischen Arbeit anhören. Anders gesehen ergibt sich jedoch ein größerer Freiraum: für verschiedene Perspektiven und größere methodische Vielfalt, die verschiedenartige und damit umfassendere Einsicht ermöglicht.

Literatur

Adám, András (1979): John von Neumann. In: (Legendi, Szentiványi 1983).

Aspray, William; Burks, Arthur (Hg.) (1987): Papers of John von Neumann on Computing and Computer Theory. Charles Babbage Institute Reprint Series for the History of Computing. Cambridge, Massachusetts 1987.

Aspray, Williams (1990): John von Neumann and the Origins of Modern Computing. Cambridge, Massachusetts 1990.

Atanasoff, John V. (1940): Computing Machine for the Solution of Large Systems of Linear Algebraic Equations. In: (Randell 1982).

Backus, John W. (1954): The IBM 701 Speedcoding System. In: Journal of the Association for Computing Machinery, Vol 1, No. 1, 1/1954.

Backus, John W. (1958): Automatic Programming. Properties and Performance of FORTRAN Systems I and II. In: Proceedings of the Symposium on the Mechanization of Thought Processes 1958.

Backus, John W. (1959): The Syntax and Sematics of the Proposed International Algebraic Language of the Zurich ACM-Gamm Conference. In: Proceedings of the International Conference on Information Processing, Paris 1959.

Backus, John W. (1980): Programming in America in the 1950s – Some Personal Impressions. In: (Meropolis et al. 1980).

Backus, John W. (1981): The History of FORTRAN I, II and III. In: (Wexelblat 1981).

Backus, John W.; Beeber R. J.; Best, Sheldon; Goldberg, Richard; Haibt, Lois M.; Herrick, Harlan L.; Nelson, R. A.; Sayre, David; Sheridan, Peter B.; Stern, H.; Ziller, Irvin (alle International Buisiness Machines Corporation, New York); Hughes, R. A. (University of California Radiation Laboratory, Livermore, California); Nutt, Roy (United Aircraft Corporation, East Hartford, Connecticut) (1957): The FORTRAN Automatic Coding System. In: Proceedings of the Western Joint Computer Conference 1957.

Backus, John W.; Herrick, Harlan (1954): IBM 701 Speedcoding and Other Automatic Programming Systems. In: Proceedings on the Symposium on Automatic Programming of Digital Computers. Washington D. C. 1954.

Balles, Ulrike (1999): Die kulturelle Erfindung des Computers – eine Analyse des Nachrichtenmagazins „Der Spiegel". Erscheint als Bericht des FB Informatik, Technische Universität Berlin.

Bauer, Friedrich L. (1959): The Problem of a Common Language, Especially for Scientific Numerical Work. In: Proceedings of the International Conference on Information Processing, Paris 1959.

Bauer, Friedrich L. (1974): Was heißt und was ist Informatik? In: IBM Nachrichten 24, Heft 223.

Bauer, Friedrich L. (1998): Wer erfand den Von-Neumann-Rechner? In: Informatik Spektrum, Band 21, Heft 2.

Bauer, Friedrich L.; Goos, Gerhard (1973): Informatik. Eine einführende Übersicht. Berlin, Heidelberg, New York 1973.

Beadle, George M. (1948): Genes and Biological Engimas. In: American Scientist, Vol. 36, No. 1, January 1948.

Becker, Barbara (1992): Künstliche Intelligenz. Konzepte, Systeme, Verheißungen. Frankfurt am Main, New York 1992.

Beckmann, F. S. (1980): Mathematical Foundations of Programming. Addison Wesley 1980.

Berr, Marie-Anne (1989): Technik und Körper. Berlin 1989.

Blum, André (1998): Der Mythos objektiver Forschung. In: DIE ZEIT, Nr. 25, 1998.

Bohnsack, Ralf (1993): Rekonstruktive Sozialforschung. Einführung in Methodologie und Praxis qualitativer Forschung. 2. Aufl., Opladen 1993.

Bohnsack, Ralf (1995): Lektion III. Interaktion und Kommunikation. In: Korte, Hermann; Schäfers, Bernhard (Hg.): Einführung in die Hauptbegriffe der Soziologie. 3. Aufl., Opalden 1995.

Bohnsack, Ralf (1996): Grupendiskussionsverfahren und Milieuforschung. In: Friberthäuser, Barbara; Prengel, Annedore (Hg.): Handbuch qualitativer Forschungsmethoden in der Erziehungswissenschaft. Weinheim, München 1996.

Bohnsack, Ralf (1997): Dokumentarische Methode. In: Hitzler, Roland; Honer, Anne (Hg.): Sozialwissenschaftliche Hermeneutik. Opladen 1997.

Bohnsack, Ralf (1998): Rekonstruktive Sozialforschung und der Grundbegriff des Orientierungsmusters. In: (Siefkes et al. 1998).

Bohnsack, Ralf (1999): Rekonstruktive Sozialforschung. Einführung in Methodologie und Praxis qualitativer Forschung. 3. Aufl., Kap. 10, Opladen 1999.

Brauer, Wilfried (1973): Die Informatik in der Bundesrepublik Deutschland. Was ist Informatik? In: GMD/DAAD (Hg.): Studien- und Forschungsführer Informatik. Bonn 1973.

Brauer, Wilfried; Münch, Siegfried (1996): Studien- und Forschungsführer Informatik. Berlin, Heidelberg, New York 1996.

Braun, Christina von (1989): NICHTICH. Logik, Lüge, Libido. Frankfurt am Main 1989.

Bremer, Robert W. (1960): Comment on COBOL. In: Management and Business Automation, März 1960.

Bruder, Klaus-Jürgen (1982): Psychologie ohne Bewußtsein. Die Geburt der behavioristischen Sozialtechnologie. Frankfurt am Main 1982.

Burks, Arthur W. (1987a): Computer Architecture and Logical Design. Introduction. In: (Aspray, Burks 1987).

Burks, Arthur W. (1987b): Biographical Notes. In: (Aspray, Burks 1987).
Burks, Arthur W. (1987c): Preface. In: (Aspray, Burks 1987).
Burks, Arthur W. (1987d): Theory of Natural and Artificial Automata. Introduction. In: (Aspray, Burks 1987).
Burks, Arthur W. (1987e): Computer Programming and Flow Diagrams. Introduction. In: (Aspray, Burks 1987).
Burks, Arthur W.; Goldstine, Herman; von Neumann, John (1946): Preliminary Discussion of the Logical Design of an Electronic Computing Instrument. 2. Auflage 1947. In: (Aspray, Burks 1987).
Changeux, J. (1984): Der neuronale Mensch. Reinbek 1984.
Chomsky, Noam (1956): Three Models for the Description of Language. In: Transactions on Information Theory, No. 2, 1956.
Chomsky, Noam (1957): Syntactic Structures. 13. Aufl., The Hague, Paris 1978.
Cortada, J. (1987): Historical Dictionary of Data Processing – Organiszations. New York 1987.
Coy, Wolfgang (1985): Industrieroboter. Zur Archäologie der zweiten Schöpfung. Berlin 1985.
Davis, M. E. (1953): The Use of Electronic Data Processing Systems in the Life Insurance Business. In: Proceedings of the AFIPS Eastern Joint Computer Conference 1953.
Davis, Martin (1995): Mathematical Logic and the Origin of Modern Computers. In: Herken, Rolf (Hg.): The Universal Turing Machine. A Half Century Survey. Wien, New York 1995.
Dijkstra, E. W. (1977): A Position Paper on Software Reliability. In: ACM SIGSOFT Software Engineering Notes 2(5), Oktober 1977.
Dittrich, Yvon (1989): Geschichte der Theorie der Formalen Sprachen. Zu den Wechselwirkungen zwischen Gesellschaft und Technik. Diplomarbeit, TH Darmstadt, Institut für Theortische Informatik, Mai 1989.
Dreyfus, Hubert L. (1985): Die Grenzen Künstlicher Intelligenz. Was Computer nicht können. Königstein/Ts. 1985.
Eckert, Presper (1980): The ENIAC (Beinhaltet den Abdruck des von ihm 1945 verfaßten „Disclosure of Magnetic Calculating Machine“). In: (Metropolis et al. 1980).
Elias, Norbert (1985): Das Credo eines Metaphysikers. Kommentare zu Poppers „Logik der Forschung“. In: Zeitschrift für Soziologie, Jg. 14, Heft 2, April 1985.
Eulenhöfer, Peter (1998a): Der Informatiker als „deus ex mathematica“. In: (Siefkes et al. 1998).
Eulenhöfer, Peter (1998b): Die formale Orientierung der Informatik. Eingereicht als Dissertationsschrift am Fachbereich Informatik der TU Berlin.

Eulemhöfer, Peter; Siefkes, Dirk; Stach, Heike; Städler, Klaus (1998): Die Konstruktion von Hybridobjekten als Orientierungsmuster der Informatik. Technische Universität Berlin, FB Informatik, Bericht 97-23.

Feyerabend, Paul (1975): Against Method. Outline of an Anarchistic Theory of Knowledge. Dtsch.: Wider den Methodenzwang. Frankfurt am Main 1986.

Fleck, James (1982): Development and Establishment in Artificial Intelligence. In: Elias, Norbert; Martins, Herminio; Whitley, Richard (Hg.): Scientific Establishments and Hierarchies. Sociology of Sciences, Vol. VI, 1982.

Fleck, Ludwik (1935): Entstehung und Entwicklung einer wissenschaftlichen Tatsache. Einführung in die Lehre vom Denkstil und Denkkollektiv. Frankfurt am Main 1994.

Floyd, Christiane (1993): Software Engineering: Kritik eines Forschungsprogramms. Manuskript zur Vorlage beim Informatik Spektrum, 31. 8. 1993.

Floyd, Christiane (1996): Autooperationale Form und situiertes Handeln. In: Hubig, Christoph (Hg.): Cognitio humana – Dynamik des Wissens und der Werte. Deutscher Kongreß für Philosophie, Leipzig, Sept. 1996.

Foucault, Michel (1974): Die Ordnung der Dinge. Frankfurt am Main 1974 (franz. 1966).

Foucault, Michel (1991): Die Ordnung des Diskurses. Frankfurt am Main 1991.

Furger, Franco; Heintz, Bettina (1998): Wahlfreiheiten. Frühe Computerentwicklung am Beispiel der Schweiz. In: (Siefkes et al. 1998).

Gabriel, Richard P.; Steele, Guy L. (1993): The Evolution of LISP. In: ACM SIGPLAN Notices, Vol. 28, No. 3, März 1993.

Galison, Peter (1997): Die Ontologie des Feindes. Norbert Wiener und die Vision der Kybernetik. In: (Rheinberger et al. 1997).

Gelernter, Herbert; Hansen, J. R.; Gerberich; C. L. (1960): A Fortran-Compiled List-Processing Language. In: Journal of the ACM, Vol. 7, 1960.

Goldstine, Adele; Goldstine Herman H. (1946): The Electronic Numerical Integrator and Computer (ENIAC). In: (Randell 1982).

Goldstine, Herman H. (1972): The Computer from Pascal to von Neumann. Princeton 1972.

Goldstine, Herman H. (1979): Die Rolle von Neumanns in der Rechentechnik. In: (Legendi, Szentivanyi 1983).

Goldstine, Herman H.; von Neumann, John (1947, 1948a, 1948b): Planing and Coding Problems for an Electronic Computing Instrument, Vol 1-3. In: (Aspray, Burks 1987).

Gorn, S. (1959): Introductory Speech (Common Symbolic Language for Computers). In: Proceedings of the International Conference on Information Processing, Paris 1959.

Gosling, James; McGilton, Henry (1995): The Java Language Environment. A White Paper. Sun Microsystems Computer Company, Oktober 1995.

Greenberger, M. (Hg.) (1962): Management and the Computer of the Future. Cambridge, Massachusetts 1962.

Hagen, Wolfgang (1989): Die verlorene Schrift. Skizzen zu einer Theorie der Computer. In: Kittler, Friedrich A.; Tholen, Christoph (Hg.): Arsenale der Seele. Literatur- und Medienanalyse seit 1870. München 1989.

Handel, Kai; Hess, Volker (1998): Sozialgeschichte der Wissenschaften – ein Abenteuer? Sherlock Holmes auf der Suche nach der eierlegenden Wollmilchau. In: (Siefkes et al. 1998).

Haraway, Donna (1981/82): The High Cost of Information in Post-World War II Evolutionary Biology. Ergonomics, Semiotics, and the Sociobiology of Communication Systems. In: The Philosophical Forum, Vol. XIII, Nos. 2-3, Winter-Spring 1981-82.

Haraway, Donna (1995): Ein Manifest für Cyborgs. Feminismus im Streit mit den Technowissenschaften. In: Haraway, Donna: Die Neuerfindung der Natur. Primaten, Cyborgs und Frauen. Frankfurt am Main, New York 1995.

Hartmanis, J. (1979): Observations about the Development of Theoretical Computer Science. In: IEEE 2/1979.

Heims, Steve J. (1987): John von Neumann and Norbert Wiener. From Mathematics to the Technologies of Life and Death. 3. Aufl., Massachusetts Institute of Technology 1987.

Heintz, Bettina (1993): Die Herrschaft der Regel. Zur Grundlagengeschichte des Computers. Frankfurt am Main, New York 1993.

Hirschberger, J. (1976): Geschichte der Philosophie. Bd. II: Neuzeit und Gegenwart. Freiburg, Bresgau 1976.

Hoffmann, Ute (1987): Computerfrauen. Welchen Anteil haben Frauen an der Computergeschichte? München 1987.

Hopper, Grace (1958): Automatic Programming. Present Status and Future Trends. In: Proceedings of the Symposium on the Mechanization of Thought Processes 1958.

Hopper, Grace; Mauchly, John W. (1953): Influence of Programming Techniques on the Design of Computers." In: Proceedings of the I. R.E., Vol. XLI, Oktober 1953.

IBM (1956): Programmer's Reference Manual. The FORTRAN Automatic Coding System for the IBM 704 EDPM. New York 1956.

IBM (1957): Programmer's Primer for FORTRAN Automatic Coding System for the IBM 704. IBM Form No. 32-0306, 1957.

Ingrassia, F. S. (1969): A Short History of Computer Programming to Date or A Programmer Strikes Back. In: Datamation 1960, S. 72.

Johnson, D. E.; Smith, R. E. (1962): FORTRAN-Autotester. A Self-Training Course Designed to Emancipate the Scientist and Engineer from the Need for the Professional Programmer. New York 1962.

Kay, Lily E. (1996): How the Genetic Code Became an Information System. Paper presented at the Dibner Systems Workshop. Massachusetts Institute of Technology 1996.

Keller, Evelyn Fox (1995): Refiguring Life. Metaphors of Twentieth-Century Biology. New York 1995.

Kettnaker, Barbara (1998): Informatik für die Schule der Zukunft. In: (Siefkes et al. 1998).

Knuth, Donald (1970): Von Neumann's First Computer Program. In: (Aspray, Burks 1987).

Knuth, Donald E.; Trabb Pardo, Luis. (1980): The Early Development of Programming Languages. In: (Meropolis et al. 1980).

König, Wolfgang (1993): Technik, Macht und Markt. Eine Kritik der sozialwissenschaftlichen Technikgeneseforschung. In: Technikgeschichte, Band 60, Heft 3, 1993.

Kraft, Peter (1977): Programmers and Managers. New York 1977.

Krämer, Sybille (1988): Symbolische Maschinen. Die Idee der Formalisierung in geschichtlichem Abriß. Darmstadt 1988.

Krämer, Sybille (1997): Kalküle als Repräsentationen. Zur Genese des operativen Symbolismus in der Neuzeit. In: (Rheinberger et al. 1997).

Kuhn, Thomas S. (1962): The Structure of Scientific Revolutions. Chicago 1962. Dtsch.: Die Struktur wissenschaftlicher Revolutionen. Frankfurt 1967.

Kutschmann, Werner (1986): Der Naturwissenschaftler und sein Körper. Die Rolle der „inneren Natur" in der experimentellen Naturwissenschaft der frühen Neuzeit. Frankfurt am Main 1986.

Lakoff, George; Johnson, Mark (1980): Metaphors We Live by. Chicago 1980.

Laubach, P.; Thompson, L. (1955): Electronic Computers: A Progress Report. In: Harvard Business Review 1955.

Lee, J. A. N. (1982): Twenty Five Years of FORTRAN. Department of Computer Science, Virginia Polytechnic Institute and State University, 82/11/12, CS82010, 1982.

Legendi, Tamás; Szentiványi, Tibor (Hg.) (1983): Leben und Werk von John von Neumann. Ein zusammenfassender Überblick. Mannheim, Wien, Zürich 1983.

Leslie, Stuart W. (1993): The Cold War and American Science. The Military-Industrial-Academic Complex at MIT and Stanford. New York 1993.

Lindner, R.; Wohak, B.; Zeltwanger, H. (1988): Planen, Entscheiden, Herrschen. Reinbek 1988.

Mahoney, Michael S. (1992): Computers and Mathematics. In: The Space of Mathematics. Philosophical, Epistemological, and Historical Explorations. Berlin, New York 1992.

Mahr, Bernd (1984): Kommentar zu von Neumanns „Die Theorie der Automaten." In: Freibeuter 21, Verlag Klaus Wagenbach, 1984.

Mauchly, John. W. (1942): The Use of High Speed Vaccuum Tube Devices for Calculating. In: (Randell 1982).

McCarthy, John (1956): The Inversion of Functions Defined by Turing Machines. In: (Shannon, McCarthy 1956).

McCarthy, John (1958). Programs with Common Sense. In: Minsky, Marvin (Hg.): Semantic Information Processing. Massachusetts Institute of Technology 1968.

McCarthy, John (1960): Recursive Functions of Symbolic Expressions. In: Horowitz, Ellis (Hg.): Programming Languages. A Grand Tour. 3. Aufl., Computer Science Press, Rockville, USA 1987.

McCarthy, John (1961): A Basis for a Mathematical Theory of Computation. In: Proceedings of the Western Joint Computer Conference, Mai 1961.

McCarthy, John (1962): Towards a Mathematical Science of Computation. In: Proceedings of the IFIP Congress, München 1962.

McCarthy, John (1981): Histroy of LISP. In: (Wexelblat 1981).

McCarthy, John; Hayes, P. J. (1969): Some Philosophical Problems from the Standpoint of Artificial Intelligence. In: Meltzer, B.; Michie, D. (Hg.): Machine Intelligence 4. Edinburgh University Press 1969.

McCorduck, Pamela (1979): Machines Who Think. A Personal Inquiry into the History and Prospects of Artificial Intelligence. San Francisco 1979.

McCulloch, Warren S.; Pitts, Walter (1943): A Logical Calculus of the Ideas Immanent in Nervous Activity. In: Anderson, James A.; Rosenfeld, Edward (Hg.): Neurocomputing. Massachusetts Institute of Technology 1988.

Mehrtens, Herbert (1990): Moderne - Sprache - Mathematik. Eine Geschichte des Streits um die Grundlagen der Disziplin und des Subjekts formaler Systeme. Frankfurt am Main 1990.

Mehrtens, Herbert (1998): Mathematik: Funktion – Sprache – Diskurs. In: (Siefkes et al. 1998).

Meropolis, N.; Howlett, J.; Rota, Gian Carlo (Hg.) (1980): A History of Computing in the Twentieth Century. Orlando 1980.

Morse, Philip M. (1977): In at the Beginnings. A Physicist's Life. Cambridge, Massachusetts 1977.

Naur, Peter (1954): The Electronic Computer and the Brain. In: (Naur 1992).

Naur, Peter (1981): The European Side of the Last Phase of the Development of ALGOL 60. In: (Wexelblat 1981).

Naur, Peter (1985): Programming as Theory Building. In: (Naur 1992).

Naur, Peter (1989): The Place of Strictly Defined Notation in Human Insight. In: (Naur 1992).

Naur, Peter (1992): Computing: A Human Activity. New York 1992.

Neumann, John von (1931): Die formalistische Grundlegung der Mathematik. In: Erkenntnis 2/1931.

Neumann, John von (1945): First Draft of a Report on the EDVAC. In: (Aspray, Burks 1987).

Neumann, John von (1948): The General and Logical Theory of Automata. In: Taub, A. H. (Hg.): John von Neumann. Collected Works. Vol. V. Oxford 1963.

Neumann, John von (1949): Theory and Organization of Complicated Automata. In: (Aspray, Burks 1987).

Neumann, John von (1952): Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components. In: (Aspray, Burks 1987).

Neumann, John von (1953): Von Neumanns Self Reproducing Automata (edited by Burks, Arthur). In: (Aspray, Burks 1987).

Neumann, John von (1958): The Computer and the Brain. New Haven 1958. Dtsch: Die Rechenmaschine und das Gehirn. München 1960.

Neumann, Klara von (1958): Vorwort. In: (von Neumann 1958).

Newell, Allen (1955): The Chess Machine. An Example of Dealing with a Complex Task by Adaption. In: Proceedings of the Western Joint Computer Concference 1955.

Newell, Allen (1963): Documentation of IPL-V. In: Communications of the ACM, VOL. 6, No. 3, März 1963.

Newell, Allen; Shaw, J. Cliff (1957): Programming the Logic Theory Machine. In: Proceedings of the Western Joint Computer Conference 1957.

Newell, Allen; Shaw, J. Cliff; Simon, Herbert A. (1957): Empirical Explorations with the Logic Theory Machine. In: Feigenbaum, Edward; Feldman, Julian (Hg.): Computers and Thought. New York 1963.

Newell, Allen; Simon, Herbert (1972): Human Problem Solving. Englewood Cliffs, New Jersey 1972.

Newell, Allen; Simon, Herbert A. (1956): The Logic Theory Machine. A Complex Information Processing System. In: IRE Transactions on Information Theory. Vol. IT-2, No. 3, September 1956.

Newell, Allen; Simon, Herbert A. (1958): Heuristic Problem Solving. The Next Advance in Operations Research. In: Operations Research, Vol. 6, Jan./Feb. 1958.

Newell, Allen; Simon, Herbert A. (1961): GPS. A Program that Simulates Human Thought. In: Feigenbaum, Edward; Feldman, Julian (Hg.): Computers and Thought. New York 1963.

Newell, Allen; Tonge, F. M. (1960): An Introduction to Information Processing Language V. In: Communications of the ACM, Vol. 3, No. 4, April 1960.

Noble, David F. (1986a): Forces of Production. San Pedro 1986.

Noble, Davis F. (1986b): Maschinenstürmer. Berlin 1986.

Pepper, Peter (1992): Grundlagen der Informatik. München 1992.

Perlis, ALan J. (1959): Proceedings of the International Conference on Information Processing, Paris 1959.

Perlis, Alan J. (1981): The American Side of the Development of ALGOL. In: (Wexelblat 1981).

Perlis, Alan J.; Samelson, Kurt (Hg.) (1958): Preliminary Report – International Algebraic Language. In: Communications of the ACM, Vol. 1, No. 12, December 1958.

Phillips, Charles A. (1960): A Common Language to Program Computers for Business Problems. Second Report. In: Computers and Automation, Oktober 1960.

Rammert, Werner (1998): Die kulturelle Orientierung der technischen Entwicklung. Eine technikgenetische Perspektive. In: (Siefkes et al. 1998).

Randell, Brian (Hg.) (1982): The Origins of Digital Computers. Selected Papers. 3. Aufl., Berlin, Heidelberg, New York 1982.

Redmond, Kent C.; Smith, Thomas M. (1980): Project Whirlwind. The History of a Pioneer Computer. Bedford, Massachusetts 1980.

Rheinberger, Hans-Jörg (1997): Von der Zelle zum Gen. Repräsentationen der Molekularbiologie. In: (Rheinberger et al. 1997).

Rheinberger, Hans-Jörg; Hagner, Michael; Wahrig-Schmidt, Bettina (Hg.) (1997): Räume des Wissens. Repräsentation, Codierung, Spur. Berlin 1997.

Rose, M. (1969): Computers, Managers and Society. Harmondsworth 1969.

Rosenbluth, Arturo; Wiener, Norbert; Bigelow, Julian (1943): Behavior, Purpose and Teleology. In: Beiheft zu Band 8 der Grundlagenstudien aus Kybernetik und Geisteswissenschaft, 1967.

Rotman, Brian (1988): Towards a Semiotics of Mathematics. In: Semiotica 72, 1988.

Rutishauser, Heinz (1956): Maßnahmen zur Vereinfachung des Programmierens. In: Elektronische Rechenmaschinen und Informationsverarbeitung. Nachrichtentechnische Fachberichte, Band 4, 1956.

Sammet, Jean E. (1961): Detailed Description of COBOL. In: Goodman, Richard (Hg.): Annual Review in Automatic Programming. Oxford, London, New York 1961.

Sammet, Jean E. (1969): Programming Languages. History and Fundamentals. Englewood Cliffs, New Jersey 1969.

Sammet, Jean E. (1981): COBOL Session. In: (Wexelblat 1981).

Schäfer, Lothar; Schnelle, Thomas (1994): Ludwig Flecks Begründung der soziologischen Betrachtungsweise in der Wissenschaftstheorie. Einleitung zu (Fleck 1935).

Scheich, Elvira (1999): Wahrheit, Natur, Wirklichkeit. Zur Veränderung erkenntnistheoretischer Perspektiven durch die Technisierung von Objektivität. In: Ritter, Martina (Hg.): Bits und Bytes von Apfel der Erkenntnis. Münster 1999.

Schmidt-Brücken, Katharina (1998): Der Rechenmaschinenspeicher als Gedächtnis. Einflüsse der Neurophysiologie auf den Rechnerbau in den vierziger Jahren. In: (Siefkes et al. 1998).

Shannon, Claude; McCarthy, John (Hg.) (1956): Automata Studies. Princeton, New Jersey 1956.

Shaw, J. Cliff; Newell, Alan; Simon, Herbert A.; Ellis, T. O. (1958): A Command Structure for Complex Information Processing. Proceedings of the Western Joint Computer Conference 1958.

Siefkes, Dirk (1993): Formale Methoden und kleine Systeme. Lernen, leben und arbeiten in formalen Umgebungen. Braunschweig, Wiesbaden 1993.

Siefkes, Dirk; Eulenhöfer, Peter; Stach, Heike; Städtler, Klaus (Hg.) (1998): Sozialgeschichte der Informatik. Kulturelle Praktiken und Orientierungen. Wiesbaden 1998.

Simon, Herbert (1945): Das Verwaltungshandeln. In: Verwaltung und Wirtschaft. Schriftenreihe der Westfälischen Verwaltungs- und Wirtschaftsakademien, Heft 12. Stuttgart, Köln 1955. Englischer Originaltitel: Administrative Behavior. New York 1945.

Simon, Herbert A. (1991): Models of my Life. The Remarkable Autobiography of the Noble Prize Winning Social Scientist and Father of Artificial Intelligence. New York 1991.

Slater, Robert (1987): Portraits in Silicon. Massachusetts Institute of Technologie 1987.

Sonntag, Michael (1988): Die Seele als Politikum. Psychologie und Produktion des Individuums. Berlin 1988.

Sonntag, Michael (1992): Historische Psychologie. In: Geschichte und Psychologie. Annäherungsversuche. Pfaffenweiler 1992.

Stach, Heike (1995): 2000 Jahre Künstliche Intelligenz? In: KI, 4/1995.

Stach, Heike (1999): Schöpfermythos Künstliche Intelligenz. Zur Strukturierung eines Technikfeldes durch die sprachliche Form. In: Ritter, Martina (Hg.): Bits und Bytes vom Apfel der Erkenntnis. Verlag Westfälisches Dampfboot, Münster.

Stoyan, Herbert (1980): LISP – Anwendungsgebiete, Grundbegriffe, Geschichte. Berlin 1980.

Stuart, W. Leslie (1993): The Cold War and American Science. The Military-Industrial-Academic Complex at MIT and Stanford. New York 1993.

Sturtevant, A. H. (1948): The Evolution and Function of Genes. In: American Scientist, Vol. 36, No. 2, April 1948.

Theißing, Florian (1994): Systemanalyse als politische Planungswissenschaft. Gesellschaftliche Rahmenbedingungen des Einsatzes in den USA der Nachkriegszeit. Studienarbeit, FB Informatik, Technische Universität Berlin, Oktober 1994.

Theißing, Florian (1995): Auf dem Weg in die Softwarekrise? Computeranwendungen und Programmentwicklung in den USA der fünfziger und sechziger Jahre. Bericht 95-14, Fachbereich Informatik, Technische Universität Berlin.

Trettin, Käthe (1991): Die Logik und das Schweigen. Zur antiken und modernen Epistemotechnik. Weinheim 1991.

Turing, Alan (1936): On Computable Numbers with an Application to the Entscheidungsproblem. In: Proceedings of the London Mathematical Society. London, 1936. Dtsch.: Über berechenbare Zahlen mit einer Anwendung auf das Entscheidungsproblem. In: Dotzler, Bernhard; Kittler, Friedrich (Hg.): Alan Turing. Intelligence Service. Berlin 1987.

Turkle, Sherry (1984): Die Wunschmaschine. Vom Entstehen der Computerkultur. Reinbek bei Hamburg 1984.

Watson, James D.; Crick, Francis H. C. (1953): Molecular Structure of Nucleic Acids. A Structure of Deoxyribonucleic Acid. In: Nature 171, 1953.

Weber, Max (1922): Wirtschaft und Gesellschaft. Tübingen 1976.

Wegner, Peter (1976): Programming Languages – The First 25 Years. In: IEEE Transactions on Computers, Vol. C-25, Dez. 1976.

Weizenbaum, Joseph (1977): Die Macht der Computer und die Ohnmacht der Vernunft. Frankfurt am Main 1977 (engl. 1976).

Weizenbaum, Joseph (1999): Interview. In: Siefkes, Dirk; Braun, Anette; Eulenhöfer, Peter; Stach, Heike; Städler, Klaus (Hg.): Pioniere der Informatik. Ihre Lebensgeschichte im Interview. Berlin, Heidelberg, New York 1999.

Wexelblat, R. L. (Hg.) (1981): History of Programming Languages. Academic Press, 1981.

Wiener, Norbert (1948): Kybernetik. Regelung der Nachrichtenübertragung im Lebewesen und in der Maschine. Düsseldorf, Wien 1963. Engl.: Cybernetics. Massachusetts Institute of Technology 1948.

Wiener, Norbert (1964): God and Golem. Massachusetts Institute of Technology 1964.

Wilkes, M. V.; Renwick, W. (1950): The EDSAC. In: (Randell 1982).

Wilkes, M. V.; Wheeler, D. J.; Gill, S. (1951): The Preparation of Programs for a Digital Computer. New York 1951.

Williams, F. C.; Kilburn, T. (1948): Electronic Digital Computers. In: (Randell 1982).

Williams, Michael R. (1985): A History of Computing Technology. Prentice-Hall Series in Computational Mathematics. London 1985.

Zemanek, Heintz (1994): Architektonische Leitideen für Computer- und Informationssysteme. In: Hellige, Hans Dieter (Hg.): Leitbilder der Informatik- und Computer-Entwicklung. artec-Paper Nr. 33, Dez. 1994.

Zemanek, Heinz (1971): Was ist Informatik? In: Elektronische Rechenanlagen, 13, Heft 4.

Zuse, Konrad (1936): Verfahren zur selbstätigen Durchführung von Rechnungen mit Hilfe von Rechenmaschinen. German patent application Z23624 (11. 4. 1936). Auszüge abgedruckt in (Randell 1982).

Der Deutsche Universitäts-Verlag

Ein Unternehmen der Fachverlagsgruppe BertelsmannSpringer

Der Deutsche Universitäts-Verlag wurde 1968 gegründet und 1988 durch die Wissenschaftsverlage Dr. Th. Gabler Verlag, Verlag Vieweg und Westdeutscher Verlag aktiviert. Der DUV bietet hervorragenden jüngeren Wissenschaftlern ein Forum, die Ergebnisse ihrer Arbeit der interessierten Fachöffentlichkeit vorzustellen. Das Programm steht vor allem solchen Arbeiten offen, deren Qualität durch eine sehr gute Note ausgewiesen ist. Jedes Manuskript wird vom Verlag zusätzlich auf seine Vermarktungschancen hin überprüft.
Durch die umfassenden Vertriebs- und Marketingaktivitäten, die in enger Kooperation mit den Schwesterverlagen Gabler, Vieweg und Westdeutscher Verlag erfolgen, erreichen wir die breite Information aller Fachinstitute, -bibliotheken, -zeitschriften und den interessierten Praktiker. Den Autoren bieten wir dabei günstige Konditionen, die jeweils individuell vertraglich vereinbart werden.

Der DUV publiziert ein wissenschaftliches Monographienprogramm in den Fachdisziplinen

Wirtschaftswissenschaft
Informatik
Kognitionswissenschaft
Sozialwissenschaft
Psychologie
Literaturwissenschaft
Sprachwissenschaft

www.duv.de

Änderungen vorbehalten.
Stand: 1.7.2000

Abraham-Lincoln-Str. 46
65189 Wiesbaden

DUV